

Vertica Documentation

Vertica Analytics Platform

Software Version: 8.1.x

Document Release Date: 1/28/2023

opentext[™]

VERTICA

Contents

Vertica® 8.1.x Documentation	4
Vertica 8.1.x Supported Platforms	11
Vertica 8.1.x New Features	37
Vertica Concepts	71
Installing Vertica	131
Getting Started	287
Administrator's Guide	351
Analyzing Data	1427
Using Flex Tables	1723
Using Management Console	1901
SQL Reference Manual	2081
Security and Authentication	3907
Extending Vertica	4025
Connecting to Vertica	4289
Using Vertica on the Cloud	4785
Integrating with Apache Hadoop	4845
Integrating with Apache Kafka	4963
Integrating with Apache Spark	5031
Vertica Pulse	5057
Best Practices for OEM Customers	5165
Vertica Plug-In for Informatica	5195
Vertica Error Messages	5241
Glossary	5413
Third-Party Software Acknowledgements	5627

Legal Notices

Warranty

The only warranties for Micro Focus International plc products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Micro Focus required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2006 - 2017 Micro Focus International plc

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Apache® Hadoop® and Hadoop are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Vertica® 8.1.x Documentation

Welcome to the Vertica 8.1.x online documentation. Published on 1/28/2023 at 1:55 AM. This PDF contains most of the content available on the My Vertica portal except for the API documentation. See the [Vertica documentation](#) page for the latest documentation.

Release Notes

You can find the latest Vertica 8.1.x Release Notes at <http://my.vertica.com/docs>.

You are viewing downloaded documentation. Vertica regularly updates the product documentation posted on the My Vertica portal. See <https://vertica.com/documentation/vertica/> for the latest documentation.

Other Resources

- [Vertica Software Downloads](#)
- [Vertica User Community](#): Technical resources for customers: Blog, forum, knowledge base, partner integration guides, troubleshooting checklists, and videos
- [The QuickStart BI Examples](#) page has technology samples powered by Vertica. Each QuickStart includes documentation and an introductory video.
- [Resources for Planning for and Configuring Hardware](#) explains how to configure hardware for Vertica.
- [Syntax Conventions](#) for Vertica documentation

Vendor Information

Micro Focus

150 CambridgePark Drive

Cambridge, MA 02140

Phone: +1 617 386 4400

E-Mail: contactvertica@microfocus.com

Web site: <http://www.vertica.com>

Syntax Conventions

These are the syntax conventions used throughout the Vertica documentation.

Syntax Convention	Description
Text without brackets or braces	Content that you type as shown, such as CREATE TABLE.
<i>table name</i>	Italics: Descriptive placeholder text that you replace with an appropriate identifier or expression.
{ param1, param2 }	Curly braces: Required parameters or arguments. For example: LIGHT_STATUS { ON OFF DIMMED} indicates that the syntax requires you to specify one value (ON, OFF, or DIMMED).
{ this that }	Vertical bar: A separator for two or more mutually exclusive items. Such items can be optional (in square brackets), or required (in curly braces). You must enter one of the given values: { ASC DESC } With square brackets, you can choose one, or neither.
[<i>param1</i>]	Square brackets: Optional statement items. For example, CREATE TABLE [<i>schema_name.</i>] <i>table_name</i> indicates that <i>schema_name.</i> is optional.
<i>column_def</i> [, ...]	Square brackets with comma and ellipses (,...): A parameter option that you can repeat. For example, <i>column_def</i> [, ...] means you can specify one or more comma-separated <i>column_def</i> options.
<i>start_json_display</i> . . . <i>end_display</i>	Vertical ellipses: Indicate text intentionally omitted for readability, such as in an example of multi-row result sets.
Indentation	Indented text: Formatting to maximize readability for

Syntax Convention	Description
	syntax sub-options. Indentation is never required, since SQL is a free-form language.

Vertica 8.1.x Supported Platforms

Welcome to Vertica Analytics Platform Supported Platforms. This document details platform support for the various components of Vertica 8.1.x.

Vertica Server and Vertica Management Console

Supported Operating Systems and Operating System Versions

Micro Focus International plc supports Vertica Analytics Platform 8.1.x running on the following 64-bit operating systems and versions on x86_x64 architecture.

Important: This document reflects what has been tested with Vertica Analytics Platform 8.1.x. Be aware that operating system vendors and communities release updates and patches for all versions of their operating systems on their own schedules. Such updates to these operating systems may or may not coincide with the release schedule for Vertica. While other versions of the operating systems listed may have been successfully deployed by customers in their environments, stability and performance of these configurations may vary. If you choose to run Vertica on an operating system version not listed in this document and experience an issue, the Vertica Support team may ask you to reproduce the issue using one of the configurations described in this document to aid in troubleshooting. Depending on the details of the case, the Support team may also ask you to enter a support ticket with your operating system vendor.

When there are multiple minor versions supported for a major operating system release, Micro Focus recommends that you run Vertica on the latest minor version listed in the supported versions list. For example, if you run Vertica on a Red Hat Enterprise Linux 7.x release, Micro Focus International plc recommends you upgrade to or be running the latest supported RHEL 7.x release, which is 7.4.

Red Hat Enterprise Linux

- Versions 6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support Red Hat Enterprise Linux 6.9 and 7.4.

Important: You cannot perform an in-place upgrade of your current Vertica Analytics Platform from Red Hat Enterprise Linux 6.6–6.9 to Red Hat Enterprise Linux 7.0/7.3/7.4. For

information on how to upgrade to Red Hat Enterprise Linux 7, see the [Upgrading Your Operating System on Nodes in Your Vertica Cluster](#). For information on changes to the operating system for Red Hat Enterprise Linux 7, see the [Red Hat Enterprise Linux 7 documentation](#).

CentOS

- Versions 6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support CentOS 6.9 and 7.4.

Important: You cannot perform an in-place upgrade of your current Vertica Analytics Platform from CentOS 6.6–6.9 to CentOS 7.0/7.3/7.4. For information on how to upgrade to CentOS 7.0/7.3/7.4, see the [Upgrading Your Operating System on Nodes in Your Vertica Cluster](#).

SUSE Linux Enterprise Server

- Version 11.0 SP3*

*Vertica 8.1.1 supports SUSE V11SP3. In the upcoming releases, Vertica will phase out support for SUSE V11SP3 and replace it with support for SUSE 12SP2. See [End-of-Support and Deprecation Notices](#) for more information.

Note: SUSE Linux Enterprise Server (SLES) 11.0 Service Pack 3 (SP3) supports the ext3 file system.

Oracle Enterprise Linux

Vertica supports the following Oracle Enterprise Linux (OEL) versions. Vertica only supports Red Hat Compatible Kernels. Vertica is not supported on unbreakable kernels (kernels with a *uel* suffix).

- Versions 6.7, 6.8, 7.3, 7.4*

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support Red Hat Enterprise Linux 6.9 and 7.4.

Debian Linux

- Versions 7.6, 7.7, 8.5, 8.9*

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support Debian Linux 8.9.

Ubuntu

- Version 12.04 LTS*
- Version 14.04 LTS

*Micro Focus is phasing out support for Vertica on Ubuntu 12.04 LTS. See [End-of-Support and Deprecation Notices](#) for more information.

Supported File Systems

Vertica Analytics Platform Enterprise Edition has been tested on all supported Linux platforms running the ext4 file system. For the Vertica Analytics Platform I/O profile, the ext4 file system is considerably faster than ext3.

While other file systems have been successfully deployed by some customers, Vertica Analytics Platform cannot guarantee performance or stability of the product on these file systems. In certain support situations, you may be asked to migrate off of these unsupported file systems to help you troubleshoot or fix an issue. In particular, several file corruption issues have been linked to the use of XFS with Vertica; Micro Focus strongly recommends not using it in production.

Vertica Analytics Platform supports Linux Volume Manager (LVM) on all supported operating systems. Your LVM version must be 2.02.66 or later, and must include device-mapper version 1.02.48 or later. For information on requirements and restrictions, see the section, [Vertica Support for LVM](#).

Supported Browsers for Vertica Management Console

Vertica Analytics Platform 8.1.x Management Console is supported on the following web browsers:

- Internet Explorer 10 and later
- Firefox 31 and later
- Google Chrome 38 and later

Vertica Server and Management Console Compatibility

Management Console (MC) 8.1.x is compatible with the latest hotfix version of Vertica server 7.2.3 and above.

Vertica 8.1.x Client Drivers

Vertica provides JDBC, ODBC, OLE DB, Python, vsql, and ADO.NET client drivers. Download the latest drivers from: [Vertica Client Drivers](#). Choose from these drivers:

- Linux and UNIX-like platforms: ODBC, JDBC, Python, and vsql clients. See [Installing the Client Drivers on Linux and UNIX-Like Platforms](#).
- Windows platforms: ODBC, ADO.NET, and OLE DB client drivers, the vsql client, the Microsoft Connectivity Pack, and the Visual Studio plug-in. See [Installing the Client Drivers and Tools on Windows](#).
- Mac OS X platforms: ODBC and vsql clients. See [Installing the Client Drivers on Mac OS X](#).
- The cross-platform JDBC client driver .jar file available for installation on all platforms.

To view a list of driver and server version compatibility, see [Vertica Analytics Platform Driver/Server Compatibility](#).

ADO.NET and OLE DB Drivers

The ADO.NET and OLE DB drivers are supported on the following platforms:

Platform	Processor	Supported Versions	.NET Requirements
Microsoft Windows	x86 (32-bit)	Windows 7 Windows 8 Windows 10	Microsoft .NET Framework 3.5 SP1 or later
Microsoft Windows	x64 (64-bit)	Windows 7 Windows 8 Windows 10	
Microsoft Windows Server	x64 (64-bit)	2012 2012 R2 2016	

JDBC Driver

All non-FIPS JDBC drivers are supported on any Java 5-compliant platform. (Java 5 is the minimum.)

The Vertica 8.1.x FIPS-enabled JDBC driver requires RHEL 6.6.

Python Driver

The Python driver requires Python version 2.7.x, and is supported on the following platforms:

Platform	Processor	Supported Versions
Red Hat Enterprise Linux	x86_64	6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*
CentOS	x86_64	6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*
SUSE Linux Enterprise	x86_64	11.0 SP3**, 12 SP2
Oracle Enterprise Linux (Red Hat Compatible Kernel only)	x86_64	6.7, 6.8, 7.3, 7.4*
Debian Linux	x86_64	7.6, 7.7, 8.5, 8.9*
Ubuntu	x86_64	12.04 LTS** 14.04 LTS

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support Red Hat Enterprise Linux 6.9, 7.4, CentOS 6.9, 7.4, Oracle Enterprise Linux 7.4, and Debian Linux 8.9.

**Micro Focus is phasing out support for this operating system. See [End-of-Support and Deprecation Notices](#) for more information.

ODBC Driver

Vertica Analytics Platform provides both 32-bit and 64-bit ODBC drivers. Vertica 8.1.x ODBC drivers are supported on the following platforms:

Platform	Processor	Supported Versions	Driver Manager
Microsoft Windows	x86 (32-bit)	Windows 7 Windows 8 Windows 10	Microsoft ODBC MDAC 2.8
Microsoft Windows	x64 (64-bit)	Windows 7 Windows 8 Windows 10	
Microsoft Windows Server	x64 (64-bit)	2012 2012 R2 2016	
Red Hat Enterprise Linux	x86_64	6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*	iODBC 3.52.6 and higher unixODBC 2.3.0 and higher DataDirect 5.3 and 6.1 and higher
CentOS	x86_64	6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*	
FIPS-compliant Red Hat Enterprise Linux	x86_64	6.6	
SUSE Linux Enterprise	x86_64	11.0 SP3*, 12 SP2	
Oracle Enterprise Linux (Red Hat Compatible Kernel only)	x86_64	6.7, 6.8, 7.3, 7.4*	
Ubuntu	x86_64	12.04 LTS**	

Platform	Processor	Supported Versions	Driver Manager
		14.04 LTS	
Debian Linux	x86_64	7.6, 7.7, 8.5, 8.9*	
AIX	PowerPC-64	7.1	
HP-UX	IA-64	11i V3**	
Solaris	SPARC-64	10	
Mac OS X	x86_64	10.10, 10.11, 10.12	

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support Red Hat Enterprise Linux 6.9, 7.4, CentOS 6.9, 7.4, Oracle Enterprise Linux 7.4, and Debian Linux 8.9.

** Micro Focus is phasing out support on this operating system. See [End-of-Support and Deprecation Notices](#) for more information.

Vertica Analytics Platform Driver/Server Compatibility

This section provides information on compatibility for the Vertica Analytics Platform driver and server versions.

Note: SHA password security is supported on client driver and server versions 7.1.x and later.

The following table indicates that, in general, all clients are forward compatible.

Client	Client Driver Version	Compatible Server Versions
All Clients	6.1.x	6.1.x, 7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x
	7.0.x	7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x
	7.1.x	7.1.x, 7.2.x, 8.0.x, 8.1.x
	7.2.x	7.2.x, 8.0.x, 8.1.x
	8.0.x	8.0.x, 8.1.x
	8.1.x	8.1.x

The following table lists FIPS 140-2 compatible clients.

Client	Client Driver Version	Compatible Server Versions
FIPS-enabled ODBC	8.0.x	8.0.x, 8.1.x
FIPS-enabled ODBC	8.1.x	8.0.x, 8.1.x
FIPS-enabled JDBC	8.1.x	8.1.x

The following table indicates that the 8.1.x ODBC client is backward compatible.

Client	Client Driver Version	Compatible Server Versions
ODBC (backwards compatibility)	8.1.x	7.1.x, 7.2.x, 8.0.x, 8.1.x

Note: Vertica Release 8.1.x and later adds backwards compatibility for the ODBC client driver. The 8.1.x ODBC client driver is backwards compatible to Vertica server version 7.1. For full compatibility with the previous server version, specify the Protocol property in your connection string. For more information about the Protocol property, see [Data Source Name \(DSN\) Connection Properties](#).

vsqI Client

The Vertica vsqI client is included in all client packages; it is not available for download separately. The vsqI client is supported on the following platforms:

Operating System	Processor
Microsoft Windows <ul style="list-style-type: none">Windows 2012, 2012 R2, 2016, all variantsWindows 7, all variantsWindows 8, all variantsWindows 10	x86, x64
Red Hat Enterprise Linux 6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*	x86, x64
CentOS 6.6, 6.7, 6.8, 6.9*, 7.0, 7.3, 7.4*	x86, x64
FIPS-compliant Red Hat Enterprise Linux 6.6	x64
Oracle Enterprise Linux 6.7, 6.8, 7.3, 7.4* (Red Hat Compatible Kernels only)	x86, x64
Ubuntu 12.04LTS**, 14.04 LTS	x86, x64
Debian Linux 7.6, 7.7, 8.5, 8.9*	x86, x64
Solaris 10	SPARC-64
AIX 7.1	PowerPC-64
HP-UX 11i V3	IA64
Mac OS X 10.10, 10.11, 10.12	x86, x64

*Vertica 8.1.1-7 and later 8.1.1 hotfixes support Red Hat Enterprise Linux 6.9, 7.4, CentOS 6.9, 7.4, Oracle Enterprise Linux 7.4, and Debian Linux 8.9.

**Micro Focus is phasing out support for these operating systems. See [End-of-Support and Deprecation Notices](#) for more information.

Perl and Python Requirements

You can use Vertica's ODBC driver to connect applications written in Perl or Python to the Vertica Analytics Platform.

Perl

To use Perl with Vertica, you must install the Perl driver modules (DBI and DBD::ODBC) and a Vertica ODBC driver on the machine where Perl is installed. The following table lists the Perl versions supported with Vertica 8.1.x.

Perl Version	Perl Driver Modules	ODBC Requirements
<ul style="list-style-type: none">• 5.8• 5.10	<ul style="list-style-type: none">• DBI driver version 1.609• DBD::ODBC version 1.22	See Vertica 8.1.x Client Drivers .

Python

To use Python with Vertica, you must install the Vertica Python Client or the pyodbc module and a Vertica ODBC driver on the machine where Python is installed. The following table lists the Python versions supported with Vertica 8.1.x:

Python Version	Python Driver Module	ODBC Requirements
2.4.6	pyodbc 2.1.6	See Vertica 8.1.x Client Drivers .
2.7.x	Vertica Python Client (Linux only)	
2.7.3	pyodbc 3.0.6	
3.3.4	pyodbc 3.0.7	

Vertica SDKs

This section details software requirements for running User Defined Extensions (UDxs) developed using the Vertica SDKs.

C++ SDK

The Vertica cluster does not have any special requirements for running UDxs written in C++.

Java SDK

Your Vertica cluster must have a Java runtime installed to run UDxs developed using the Vertica Java SDK. Micro Focus has tested the following Java Runtime Environments (JREs) with this version of the Vertica Java SDK:

- Oracle Java Platform Standard Edition 6 (version number 1.6)
- Oracle Java Platform Standard Edition 7 (version number 1.7)
- Oracle Java Platform Standard Edition 8 (version number 1.8)
- OpenJDK 6 (version number 1.6)
- OpenJDK 7 (version number 1.7)
- OpenJDK 8 (version number 1.8)

Python SDK

The Vertica Python SDK does not require any additional configuration or header files.

R Language Pack

The Vertica R Language Pack provides version 3.2.5 of the R runtime and associated libraries for interfacing with Vertica. You install the R Language Pack on the Vertica server.

FIPS 140-2 Supported Platforms

Micro Focus Vertica uses a certified OpenSSL FIPS 140-2 cryptographic module to meet the security standards set by the National Institute of Standards and Technology (NIST) for Federal Agencies in the United States or other countries.

FIPS-enabled Vertica requires the following:

- Red Hat Linux 6.6, a FIPS compliant operating system
- OpenSSL 1.0.1e
- A user-generated certificate signed by an approved Certificate Authority
- TLS 1.2 to support the server-client connection for a FIPS-enabled system

Supported Drivers

Vertica supports the following client drivers for FIPS-compliance:

- vsql
- ODBC
- JDBC

Important: FIPS-enablement is not supported in the Management Console.

For more information see [Federal Information Processing Standard](#).

Vertica Integrations for Hadoop

Micro Focus has tested Vertica 8.1.x with the following Hadoop distributions. Micro Focus expects Vertica to work with subsequent Hadoop distributions, and tests such later distributions as soon as practical.

Distribution	Versions Tested with Vertica Server 7.0.x
Cloudera (CDH)	<ul style="list-style-type: none">• 5.7• 5.8
HortonWorks Data Platform (HDP)	<ul style="list-style-type: none">• 2.4• 2.5
MapR	<ul style="list-style-type: none">• 5.1• 5.2

You must apply patches for the following issues: HDFS-8855, HDFS-8696, HDFS-7280, HDFS-7279, HDFS-7270, and HDFS-7945. See your Hadoop vendor documentation for further instructions.

Packs, Plug-Ins, and Connectors for Partner Products

Micro Focus provides the following optional module for Vertica client machines.

Informatica PowerCenter Plug-In

The Vertica plug-in for Informatica PowerCenter is supported on the following platforms:

Plug-in Version	Informatica PowerCenter Versions	Vertica Versions
1.1	9.x	6.x (limited functionality) 7.x (all enhancements) 8.x (all enhancements)

For a complete list of supported client drivers, see [Vertica 8.1.x Client Drivers](#).

For more information about the Informatica PowerCenter Plug-in, see [Introduction to Using Informatica PowerCenter with Vertica](#).

Vertica on Amazon Web Services

Micro Focus provides a preconfigured AMI for users who want to run Vertica Analytics Platform on Amazon Web Services (AWS). This AMI allows users to configure their own storage and has been configured for and tested on AWS. This AMI is the officially supported version of Vertica Analytics Platform for AWS.

Note that Micro Focus develops AMIs for Vertica on a slightly different schedule than our product release schedule. Therefore, AMIs will be available for Vertica releases sometime following the initial release of Vertica software.

Vertica in a Virtualized Environment

Vertica runs in the following virtualization environment:

Important: Vertica does not support suspending a virtual machine while Vertica is running on it.

Host

- VMware version 5.5
- The number of virtual machines per host did not exceed the number of physical processors
- CPU frequency scaling turned off at the host level and for each virtual machine
- VMware parameters for hugepages set at version. 5.5 defaults

Input/Output

- Measured by [vioperf](#) concurrently on all Vertica nodes When running vioperf, provide the `–duration=2min` option and start on all nodes concurrently
- 25 megabytes per second per core of write
- 20+20 megabytes per second per core of rewrite
- 40 megabytes per second per core of read
- 150 seeks per second of latency (SkipRead)
- Thick provisioned disk, or pass-through-storage

Network

- Dedicated 10G NIC for each Virtual Machine
- No oversubscription at the switch layer, verified with [vnetperf](#)

Processor

- Architecture of Sandy Bridge (HP Gen8 or higher)
- 8 or more virtual cores per virtual machine
- No oversubscription
- [vcupperf](#) time of no more than 12 seconds (\approx 2.2 GHz clock speed)

Memory

- Pre-allocate and reserve memory for the VM
- 4G per virtual core of the virtual machines

HP has tested the configuration above. While other virtualization configurations may have been successfully deployed by customers in development environments, performance of these configurations may vary. If you choose to run Vertica on a different virtualization configuration and experience an issue, the Vertica Support team may ask you to reproduce the issue using the configuration described above, or in a bare-metal environment, to aid in troubleshooting. Depending on the details of the case, the Support team may also ask you to enter a support ticket with your virtualization vendor.

Vertica Integration for Apache Kafka

You can use Vertica with the Apache Kafka message broker. For more information on Kafka integration, refer to [How Vertica and Apache Kafka Work Together](#).

Kafka Versions

Vertica supports the following Kafka distributions:

Apache Kafka Versions	Vertica Versions
0.9.0	7.2 SP2 and later
0.10	8.1 and later

Java Versions

The data streaming job scheduler needs to make a JDBC connection to the target database, and requires Java 7.0 or later to run.

Vertica Support for LVM

Vertica Analytics Platform 8.1.x supports Linux Volume Manager (LVM) on all supported operating systems.

LVM Version Supported

Vertica Analytics Platform supports LVM version 2.02.66 or later, and must include device-mapper version 1.02.48 or later.

LVM Configuration Notes

In configuring LVM:

- When you create logical volumes with the `lvcreate` command, use the `readahead` option to set the read ahead sector count to greater than 2048 KB.
- You can use the default settings for all other LVM options.

LVM Restrictions

The following limitations apply to LVM support:

- Your logical volume file system type must be ext4.

Note: SUSE Linux Enterprise Server (SLES) 11.0 Service Pack 3 (SP3) supports the ext3 file system. However, Vertica is phasing out support for this operating system. See [End-of-Support and Deprecation Notices](#) for more information.

- You cannot have physical drives shared across several nodes.
- Vertica supports linear logical volumes only. Vertica does not support striped or mirrored logical volumes.

- Vertica supports extending logical volumes (`lvextend`), but not reducing the size of a logical volume.
- Vertica recommends frequent backups.
- Vertica does not support LVM backup and restore, such as LVM snapshot and merge. Use the Vertica backup file `vbr.py`.
- Vertica does not support LVM space reclamation. Space reclamation is duplicated with reducing the size of a logical volume.
- Vertica does not support LVM migration. Use Vertica Copy operations.
- Vertica does not support LVM high availability. Use Vertica high availability capabilities.
- Vertica does not support LVM RAID .Configure RAID at the disk controller level.

Vertica Integration for Apache Spark

You can use the Vertica Connector for Apache Spark to transfer data between Vertica and Apache Spark. Vertica supports the following Apache Spark versions:

Apache Spark Version	Scala Version	Vertica Versions
1.6	2.10	8.0.x, 8.1.x
2.0	2.10, 2.11	8.0.x, 8.1.x
2.1	2.10, 2.11	8.0.x, 8.1.x

Note: Vertica 8.0.x is compatible with Spark versions 2.0 and 2.1 and Scala versions 2.10 and 2.11 when you use the Spark connector distributed with Vertica version 8.1.x.

For more information on Apache Spark integration, refer to [Integrating with Apache Spark](#).

End-of-Support and Deprecation Notices

These end-of-support notices apply to specific client, Linux, Hadoop, and Kafka distributions.

End-of-Support Notices

There are no new supported platforms end of support notices in this release of Vertica. However, Vertica is phasing out support for certain client platforms and server distributions. See [Deprecation Notices](#).

Deprecation Notices

Vertica is phasing out support for the following clients on the listed platforms.

Client	End of Support for Platform
All Client Drivers	Ubuntu 12.04 LTS
	SUSE 11 SP3

For a list of supported platforms for Vertica clients, see [Vertica 8.1.x Client Drivers](#).

Vertica is phasing out support for the following Linux distributions.

Distribution	End of Support for Version
Ubuntu	12.04 LTS
SUSE	11 SP3

For a list of supported server operating systems, see [Vertica Server and Vertica Management Console](#).

Vertica 8.1.x New Features

Welcome to Vertica Analytics Platform New Features. This guide briefly describes the new features introduced in the most recent releases of Vertica and provides references to detailed information in the documentation set.

To see a list of known and fixed issues in the most current release, see the Vertica Release Notes, which you can find at https://my.vertica.com/docs/ReleaseNotes/8.0.x/Vertica_8.1.x_Release_Notes.htm.

Vertica 8.1.1 New Features and Changes

Read the topics in this section for information about new and changed functionality in Vertica 8.1.x.

Supported Platforms

This section contains information on updates to supported platforms for Vertica Analytics Platform 8.1.x.

- [Changes for Operating System Support](#)
- [Changes for Vertica Integration with Apache Spark](#)

Changes for Operating System Support

Vertica operating system support changes follow.

Server

Vertica has added support for Oracle Enterprise Linux (OEL) 6.8. For a list of all supported operating systems, see [Vertica Server and Vertica Management Console](#).

Vertica has added support for Linux Volume Manager on all supported operating systems. For information on supported file systems, See [Vertica Server and Vertica Management Console](#). For information on requirements and restrictions, see the section, [Vertica Support for LVM](#).

Vertica has deprecated support for Ubuntu 12.04 LTS and SUSE 11 SP3. For more information, refer to [End-of-Support and Deprecation Notices](#).

Client

For all client drivers, Vertica has added support for OEL 6.8 and SUSE 12 SP2.

For a complete list of client drivers, see [Vertica 8.1.x Client Drivers](#).

Vertica has deprecated support for Ubuntu 12.04 LTS and SUSE 11 SP3 for all client drivers. For more information, refer to [End-of-Support and Deprecation Notices](#).

Changes for Vertica Integration with Apache Spark

Vertica has added support for Apache Spark version 2.1.

For a list of supported Spark versions, see [Vertica Integration for Apache Spark](#).

For More Information

See [Vertica 8.1.x Supported Platforms](#)

Upgrade and Installation

Model Schema Restrictions

As of this release, you can create models only in `public` and user-created schemas (excluding `HCatalog`). Attempts to create models elsewhere return an error. If the upgrade finds a model that was created in a schema that does not conform with this restriction, it moves the model to the `public` schema. If another object of the same name already exists in `public`, the upgrade renames the moved model as follows:

```
model-name >> model-name_vN
```

where N is an integer ≥ 0 . If *model-name_vN* identifies an object that already exists in the schema, Vertica increments N until it finds a name that is unique in the schema. After the upgrade is complete, check the Vertica log for all changes of this type.

Database Management

This section contains information on updates to database operations for Vertica Analytics Platform 8.1.x.

Map New IP Addresses

Vertica provides an easy way to apply new IP addresses to nodes in a cluster in cases where old IP addresses are no longer valid. For example, if you are running Vertica in a cloud environment and the cloud vendor reassigns IP addresses.

Mapping new IP addresses involves creating a mapping file and running commands in admintools. For more information see [Mapping New IP Addresses](#).

For More Information

See [Managing the Database](#).

Data Analysis

This section contains information on updates to data analysis for Vertica Analytics Platform 8.1.x.

Machine Learning for Predictive Analytics

Machine Learning for Predictive Analytics new features include:

- In-database predictive modeling for classification problems using random forest
- In-database predictive modeling for regression problems using SVM (support vector machine)
- Support for the ability to detect outliers by group
- Support for the ability to extract model attributes
- Support for imbalanced data process using a hybrid method of under-sampling and over-sampling

For More Information

See [Machine Learning for Predictive Analytics](#).

Client Connectivity

This section contains information on updates to connection information for Vertica Analytics Platform 8.1.x.

Direct ODBC or OLE DB Log Entries to ETW

You can now have Vertica send ODBC or OLE DB log entries to Event Tracing for Windows (ETW).

To enable this feature, you add a new string to your Windows Registry, LogType, and you set the string value to ETW. Once set, ODBC or OLE DB log entries appear in the Windows Event Viewer.

When you turn this feature on, be aware of the differences between the LogLevel parameter settings and how those settings are compressed for the Windows Event Viewer. LogLevel settings are 0 through 6 (described in [Additional Parameter Settings](#)). Entries are sent to the Windows Event Viewer as four levels rather than six.

For more detailed information about this feature for ODBC, see [Register the ODBC Driver as a Windows Event Log Provider, and Enable the Logs](#)

For more detailed information about this feature for OLE DB, see [Register the OLE DB Driver as a Windows Event Log Provider, and Enable the Logs](#).

Disable Local Copying in JDBC Connections

The JDBC connector now has the setting DisableCopyLocal. When set to true, it disables file-based COPY LOCAL operations, including copying data from local files and using local files to store data and exceptions. For more information, refer to [JDBC Connection Properties](#).

For More Information

See [Connecting to Vertica](#).

Workload Management

This section contains information on updates to workload management for Vertica Analytics Platform 8.1.x.

Configuring a Grace Period to Handle Session Socket Blocking

A session socket can occasionally be blocked indefinitely while awaiting client input or output for a given query. You can now set a grace period to handle session socket blocking, at the session, user, node, and database levels. If a socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated.

For details, see [Handling Session Socket Blocking](#) in the Administrator's Guide.

For More Information

See [Managing Workloads](#).

Geospatial Analytics

This section contains information on updates to Vertica Geospatial Analytics for Vertica Analytics Platform 8.1.x.

Improved STV_NN Analytic Performance

The Geospatial Analytic function [STV_NN](#) is now at least twice as fast when performing point-to-point calculations.

For More Information

See [Geospatial Analytics](#).

Backup, Restore, Recovery, and Replication

This section contains information on updates to backup and restore operations for Vertica Analytics Platform 8.1.x.

Backup and Restore to Amazon S3

Vertica now has the ability to backup and restore to Amazon S3 Standard for Vertica instances hosted locally and on Amazon Web Services. For more information, refer to [Creating Backups on Amazon S3](#).

Sample VBR Configuration Files

Vertica now includes sample configuration files that you can copy, edit, and deploy for your various vbr tasks. Vertica automatically installs these files at `/opt/vertica/share/vbr/example_configs`. For more information, refer to [Sample VBR .ini Files](#).

Ability to View All Backups in a Location

The Vertica backup and restore utility now has the following additional parameters for the `listbackup` task:

- `--list-all` - Lists all backups store on the hosts and paths listed in the specified configuration file.
- `--json` - Displays a JSON delimited list of all backups stored on the hosts and paths listed in the specified configuration file.
- `--list-output-file` - Outputs a file containing a JSON delimited list of all backups stored on the hosts and paths listed in the specified configuration file.

For more information, refer to [Viewing Backups](#).

For More Information

See [Backing Up and Restoring the Database](#).

Table Data Management

This section summarizes new options and changes in Vertica Analytics Platform 8.1.x, for managing table data in a Vertica database.

Projections and Anchor Tables Required to Share Same Schema

As of release 8.1.1, Vertica requires that all projections of a given table must share the same schema. On upgrade to this version, Vertica checks whether all projections are in the same schema as their respective anchor tables. If not, Vertica moves the projections to the appropriate schema, resolving name conflicts as needed.

Support will continue for specifying schemas in statements and functions, to ensure backward compatibility with existing scripts.

New Header_Names Parameter for fcsvparser

The fcsvparser has a new `HEADER_NAMES= ' '` parameter. Use this optional parameter to override existing column names, or to add column names where they are missing. For example, to add four column headings when loading a CSV file that has none, use a definition such as this with the new parameter:

```
HEADING_NAMES= 'NAME , CITY , STATE , ZIP '
```

For more information and examples, see [Loading CSV Data](#) and [FCSVPARSER](#)

For More Information

See [Managing Tables](#), [Creating Flex Tables](#), and [Understanding Flex Tables](#)

Management Console

This section contains information on updates to the Management Console for Vertica Analytics Platform 8.1.x.

Run Queries Through MC

You can now use MC to run SQL queries on your database. Enter ad hoc queries, upload a SQL script, or select previously run queries using the MC Query Runner to get query results through your browser.

To reach the Query Runner, select your database from the Home page or the Databases and Clusters page to view your database's Overview page. Select **Query Execution** at the bottom of the Overview page.

For more about the Query Runner, see [Running Queries in Management Console](#)

The screenshot shows the Management Console interface for the Query Runner. The breadcrumb navigation is "Databases and Clusters > VMart > Query Runner". On the left, there is a "Query History" panel with a "Clear all" button and a "Filter previous queries" input. Below this, several query snippets are listed. The main area contains a SQL query editor with the following code:

```
1 SELECT sales_quantity, sales_dollar_amount, transaction_type, cc_name
2 FROM online_sales.online_sales_fact
3 INNER JOIN online_sales.call_center_dimension
4 ON (online_sales.online_sales_fact.call_center_key
5     = online_sales.call_center_dimension.call_center_key
6     AND sale_date_key = 156)
7 ORDER BY sales_dollar_amount DESC;
8 SELECT order_number, date_ordered
9 FROM store.store_orders_fact orders
10 WHERE orders.store_key IN (
11   SELECT store_key
12   FROM store.store_dimension
13   WHERE store_state = 'MA')
14   AND orders.vendor_key NOT IN (
15     SELECT vendor_key
16     FROM public.vendor_dimension
17     WHERE vendor_state = 'MA')
18   AND date_ordered < '2012-03-01';
19 SELECT store_key, order_number, date_ordered
```

Below the query editor is a green "Execute Queries" button. Underneath, there are tabs for "Query Results", "Query Plan", and "Query Profile". The "Query Results" tab is active, showing a table with the following data:

sales_quantity	sales_dollar_amount	transaction_type	cc_name
7	589	purchase	Central Midwest
8	589	purchase	South Midwest
8	589	purchase	California

At the bottom of the results table, it says "2514 rows | Execution time: 0.113s". The bottom navigation bar includes "Overview", "Activity", "Manage", "Design", "Load", "Query Execution" (highlighted), "Query Plan", "License", and "Settings".

Explore New Features After Upgrade

MC displays a new "Explore New Features in Vertica" notification after upgrade. Through this you can explore the features in the newest version of Vertica, the Vertica YouTube channel for new features highlights and video tutorials, and the deprecated functionality list.

You will see this notification upon first upgrading and logging into MC.

Improved Contextual Help

The contextual help in MC now has a new look and feel, and provides a new menu you can use to navigate to online documentation, Vertica video tutorials, and the Vertica forum.

Click the question mark icon at the top right of any page in MC to view contextual help.

For More Information

See [Using Management Console](#) .

SQL Functions and Statements

This section contains information on updates to SQL functions and statements for Vertica Analytics Platform 8.1.x:

MERGE Support for Subqueries and Views as Source Data

MERGE functionality has been extended to support specifying views and subqueries as merge source data. The MERGE statement 's USING clause can now:

- Specify a view in the same way as a table. Vertica expands the view name to the query that it encapsulates, and uses the result set as the merge source data.
- Specify a subquery. Vertica executes the query and uses the result set as the merge source data.

For details, see [MERGE Source Options](#) in the Administrator's Guide.

Shortcut Constraint DEFAULT USING

Before this release, a column was required to specify separate subqueries for DEFAULT and SET USING constraints, even if both constraints specified the same subquery. Now, you can combine the two constraints and specify their subquery once, as follows:

```
column-name data-type DEFAULT USING (subquery)
```

For example, the following two statements are equivalent:

```
=> ALTER TABLE T1 ADD yy varchar(20)  
    DEFAULT (SELECT y FROM T2 WHERE (T1.c1 = T2.c1))  
    SET USING (SELECT y FROM T2 WHERE (T1.c1 = T2.c1));
```

```
=> ALTER TABLE T1 ADD yy varchar(20) DEFAULT USING (SELECT y FROM T2 WHERE (T1.c1 = T2.c1));
```

Note: When you use DEFAULT USING, the DDL that Vertica generates for the table specifies separate DEFAULT and SET USING constraints.

Forcing Early Materialization

You can qualify one or more tables in a query with the hint [EARLY_MATERIALIZATION](#). This hint can be useful in cases where late materialization of join inputs precludes other optimizations—for example, pushing aggregation down the joins or using live aggregate projections.

For More Information

See [SQL Reference Manual](#).

System Table Updates

This section contains information on system tables updates for Vertica Analytics Platform 8.1.x.

GRACE_PERIOD Column

System tables [SESSIONS](#) and [USERS](#) now include a GRACE_PERIOD column that is set by [SET SESSION GRACEPERIOD](#) and [CREATE USER/ALTER USER](#), respectively. For general information, see [Handling Session Socket Blocking](#) in the Administrator's Guide.

Table for Auditing External Tables

System table [EXTERNAL_TABLE_DETAILS](#) records information about external tables. You can use this table to audit the amount of ORC and Parquet data you are using, which might be limited by your license.

Table for Monitoring Memory Pool Allocations

System table [ALLOCATOR_USAGE](#) provides information on memory pool allocation and usage for each Vertica node. You can use the information presented in the table to help diagnose memory pool issues.

Table for Monitoring Notifier Errors

The system table [NOTIFIER_ERRORS](#) now provides information on errors encountered by [notifiers](#).

For More Information

See [Vertica System Tables](#).

Apache Hadoop Integration

This section contains information on updates to Hadoop-integration information for Vertica Analytics Platform 8.1.x.

Exporting Parquet Data

You can export data from Vertica, either to share it with other Hadoop-based applications or to move lower-priority data from ROS to less-expensive storage. The [EXPORT TO PARQUET](#) statement exports a result set as Parquet data. After exporting ROS data, you can drop affected ROS partitions to reclaim storage space. If you need to access the data in Vertica again, you can create external tables from the exported data.

For more information, see [Exporting Data](#)

HCatalog Connector Uses HiveServer2

Previously the HCatalog Connector read Hive metadata using the WebHCat web service, which has poor performance. The HCatalog Connector now supports HiveServer2, which has a faster JDBC interface, and uses it by default. (You can still use WebHCat if you prefer it.)

When using [CREATE HCATALOG SCHEMA](#) to create a schema in Vertica to mirror a Hive schema, you can now specify an additional parameter, `HIVESERVER2_HOSTNAME`. This statement has an additional (optional) parameter, `WEBHDFS_ADDRESS`. The `WEBHDFS_PORT` parameter has been removed.

For more information, see [Using the HCatalog Connector](#).

Cloudera Manager Integration

The Cloudera distribution of Hadoop includes Cloudera Manager, a web-based tool for managing a Hadoop cluster. Cloudera Manager can manage any service for which a service description is available. This release includes a service description for Vertica. After installing this service description, you can manage your Vertica cluster from the Cloudera Manager console. You can start, stop, and add nodes and manage memory pools.

For more information, see [Integrating With Cloudera Manager](#).

hdfs Scheme Fully Supports High Availability Name Node

Previous versions of Vertica added support for using the `hdfs` URL scheme in HDFS clusters with High Availability Name Node (HA NN). Reads and writes using the `hdfs` scheme can sometimes fall back to using `webhdfs`, which does not support HA NN. Vertica now supports

HA NN when falling back to webhdfs from hdfs. For more information, see the description of the LibHDFS++ FAILOVER RETRY event in [Check Query Events](#) in Analyzing Data.

For More Information

See [Integrating with Apache Hadoop](#)

Apache Kafka Integration

Vertica version 8.1.1 includes the following changes to its integration with Apache Kafka.

Setting a Starting Offset for a Microbatch

Previously, a microbatch always started loading data from the start of a stream. You can now use the `--offset` parameter to set an offset in the stream where the microbatch will start loading data. This new parameter lets you have the microbatch skip older data in a stream or to force it reload old data. See [Microbatch Utility Options](#) for details.

Setting an Ending Offset for KafkaSource

Previously, KafkaSource could end streaming either after a timeout or on reaching the end of a stream. You can now have it end streaming when it reaches a specific offset in the stream. See [Using COPY with Data Streaming](#) for more information.

For More Information

See [Integrating with Apache Kafka](#)

Apache Spark Integration

This section contains updates to the Apache Spark integration for Vertica Analytics Platform version 8.1.x.

Support for Spark 2.1

The Vertica Connector for Apache Spark now supports Spark 2.1. The connector's JAR file is compatible with a specific combination of Spark and Scala versions. Micro Focus supplies multiple JAR files, one for each supported version combination. See [Getting the Spark Connector](#) in [Integrating with Apache Spark](#) for more information.

For More Information

See [Integrating with Apache Spark](#) .

Performance Improvements

This section summarizes performance improvements in Vertica Analytics Platform 8.1.x.

Common Subexpression Elimination

Queries with complex expressions are routinely generated by business intelligence tools, and can occasionally be slow to execute. Query execution can be optimized by identifying and reusing common subexpressions (*common subexpression elimination*, or CSE). With this release, Vertica provides its own CSE implementation, so evaluation results between expressions can be shared within a single query.

All subexpressions in an expression are candidates for substitution. Substitution does not apply to trivial substitutions—for example, variables and constants—and to expressions that occur only once. Substitution can be recursive—that is, substitution can be applied to substituted expressions.

Better Handling of Concurrent Queries

Vertica now handles throughput of concurrent queries much more efficiently, yielding significant improvements in execution time.

Execution of Queries on Tables with Many Projections

With this release, the number of projections anchored to a table has significantly less impact on the execution time of queries on that table.

Vertica Community Edition VM

This section contains information on updates to the Vertica Community Edition VM for Vertica Analytics Platform 8.1.x.

VM Enhancements

The Vertica Community Edition VM, previously called the Vertica Virtual Machine, is significantly enhanced for usability in 8.1 SP1. Vertica Community Edition is provided in a VM so that users can quickly gain hands-on experience with Vertica.

Prior to 8.1 SP1, the VM came with Vertica Community Edition installed, but setting up a learning environment was left to the user. For example, the user had to create a database, generate and load VMart example data, and then learn how to use Vertica Administration Tools and vsql.

With 8.1. SP1, the Vertica Community Edition VM (Vertica CE VM) is a live, pre-configured environment. The Vertica client tools, including Management Console, are connected to the VMart example database, and a user guide provides step-by-step exercises for using the tools. Many of the exercises introduce the user to Vertica Management Console, a user-friendly tool that runs in a browser. The Vertica CE VM also includes the complete Vertica documentation set.

For More Information

See [Downloading and Starting the Vertica Community Edition VM](#)

Vertica 8.1 New Features and Changes

Read the topics in this section for information about new and changed functionality in Vertica 8.1.x.

Supported Platforms

This section contains information on updates to supported platforms for Vertica Analytics Platform 8.1.x. Administrator's Guide

- [Changes for Operating System Support](#)
- [Changes for Hadoop and Kafka Distribution Support](#)

Changes for Operating System Support

Vertica operating system support changes follow.

Server

Vertica has added support for these operating systems:

- Red Hat Enterprise Linux (RHEL) 7.3
- CentOS 7.3
- Oracle Enterprise Linux (OEL) 7.3

For a list of all supported operating systems, see [Vertica Server and Vertica Management Console](#).

Vertica has added support for Linux Volume Manager on RHEL 6.8. For information on supported file systems, See [Vertica Server and Vertica Management Console](#). For information on requirements and restrictions, see the section, [Vertica Support for LVM](#).

Vertica has ended support for RHEL 6.5 and CentOS 6.5. For more information, refer to [End-of-Support and Deprecation Notices](#).

Client

Vertica has added support for the following:

- For the ADO.NET, OLE DB, and ODBC clients, Vertica now supports Windows Server 2016.
- For the ODBC and vsql clients, Vertica now supports AIX 7.1 and Mac OS X 10.12.
- Vertica has added a FIPS-compliant (FIPS 140-2) JDBC client driver.

For a complete list of client drivers, see [Vertica 8.1.x Client Drivers](#).

For information on FIPS 140-2 prerequisites, see [FIPS 140-2 Supported Platforms](#).

Vertica has ended support for AIX 5.3, AIX 5.6, and Mac OS X 10.9. For more information, refer to [End-of-Support and Deprecation Notices](#).

ODBC Client Driver Backwards Compatibility

In general, all Vertica clients are forward compatible with later Vertica server versions. For Release 8.1.x, Vertica has added backwards compatibility for the ODBC client driver.

- The 8.1.x ODBC client driver is backwards compatible to Vertica server version 7.1.
- For full compatibility between the 8.1.x ODBC client driver and a previous Vertica server version, Vertica has added a connection string property named Protocol. The Protocol connection string option specifies the front-end protocol that the client and server use to communicate. For more information about the Protocol connection string property, see [Data Source Name \(DSN\) Connection Properties](#).

Changes for Hadoop and Kafka Distribution Support

Vertica has added support for the following:

- Hortonworks Data Platform (HDP) 2.5
- MapR 5.2
- Apache Kafka 0.10

For a list of supported Hadoop distributions, see [Vertica Integrations for Hadoop](#).

For a list of supported Apache Kafka distributions, see [Vertica Integration for Apache Kafka](#).

Vertica has ended support for Cloudera CDH 5.6, Hortonworks HDP 2.3, MapR 4.1 and 5.0, and Apache Kafka 0.8. For more information, refer to [End-of-Support and Deprecation Notices](#).

More Details

For complete information on platform support see [Vertica 8.1.x Supported Platforms](#)

Performance Improvements

This section summarizes performance improvements in Vertica Analytics Platform 8.1.0.

Queries with Multiple Distinct Aggregates

Queries that include multiple distinct aggregates now execute more efficiently. The scale of this improvement is especially significant for queries where all distinct aggregate columns have a similar number of distinct values.

Query Optimization

This section contains information on query execution improvements for Vertica Analytics Platform 8.1.x.

Flattened Tables

Before release 8.1., Vertica users could denormalize their data by combining all fact and dimension table columns in a single 'fat' table. However, this approach required them to maintain redundant sets of normalized and denormalized data.

Release 8.1. addresses this issue with flattened tables, which include columns that get their values by querying other tables. Operations on the source tables and flattened table are decoupled: changes in one are not automatically propagated to the other. This minimizes the overhead that is otherwise typical of conventional denormalized tables.

For details, see [Flattened Tables](#) in Analyzing Data.

Wide Column Data Queried for Insertion into Another Table

Query execution is significantly improved in two cases where a table is populated with variable length columns (VARCHAR/LONG VARCHAR and VARBINARY/LONG VARBINARY) from another table:

- [CREATE TABLE...AS SELECT](#)
- [INSERT...SELECT](#)

If the source and target tables share the same sort order but are segmented differently, Vertica buffers the query results. With release 8.1.x, Vertica optimizes buffering for wide columns, so overall query execution now incurs much less overhead than before.

Note: Performance improvements can vary, depending on the width of the source column data.

Queries with Multiple Distinct Aggregates

Vertica has improved the performance of queries that include multiple distinct aggregates. While all queries that include multiple distinct aggregates experience this improvement, the scale of the performance improvement depends upon your data and how your queries are structured. Queries where all the distinct aggregate columns have a similar number of distinct values experience the greatest performance improvement. Queries where the distinct aggregate columns have a vastly different number of distinct values experience a performance improvement to a lesser extent.

Security and Authentication

This section contains information on updates to security and authentication features for Vertica Analytics Platform 8.1.x.

Encrypting Passwords on ODBC DSN

You can encrypt the database password and store it in registry. This prevents illicit access to the database.

For more information see [Encrypting Passwords on ODBC DSN](#).

More Details

For more information see [Security and Authentication](#) .

Data Analysis

This section contains information on updates to data analysis for Vertica Analytics Platform 8.1.x.

Machine Learning for Predictive Analytics

Machine Learning for Predictive Analytics new features include:

- In-database predictive modeling for classification problems, using support vector machine (SVM)
- New normalization functions to save transformation parameters
- An impute function to impute missing values in your data
- Support for [ALTER MODEL](#) and [DROP MODEL](#) for machine learning models

More Details

For more information see [Analyzing Data](#)

Client Connectivity

This section contains information on updates to connection information for Vertica Analytics Platform 8.1.x.

\timing Meta-command

You can toggle the `\timing` command from Vsql as follows:

```
=> \timing on  
Timing is on
```

```
=> \timing off  
Timing is off
```

You can also enable `/timing` from the command line with the `-i` command:

```
$VSQL -h host1 -U user1 -d VMart -p 15 -w ***** -i -f transactions.sql
```

For more information see [\timing](#) and [Command-Line Options](#).

More Details

For more information see [Connecting to Vertica](#).

Query Management

This section contains information on updates to Query Management for Vertica Analytics Platform 8.1.x.

New Semantics for Complex Joins

With this release, Vertica introduces new semantics that allow directed queries to support complex half joins and cross joins. These semantics can also be used to construct queries that filter the results of half-join subqueries.

More Details

See [Half Join and Cross Join Semantics](#) in the Administrator's Guide.

Tables

This section summarizes the Vertica Analytics Platform 8.1.x updates to regular and flex tables.

Creating a Flex Table from Query Results

The CREATE FLEX TABLE AS statement allows a user to create a flex table from the results of a query. This statement was already functional for regular tables--this feature enables it for flex tables as well.

More Details

For more information see [Managing Tables](#), [Creating Flex Tables](#), and [Understanding Flex Tables](#)

Geospatial Analytics

This section contains information on updates to Vertica Geospatial Analytics for Vertica Analytics Platform 8.1.x.

GeoHash Support

Vertica supports GeoHashes. A GeoHash is a geocoding system for hierarchically encoding increasingly granular spatial references. Vertica supports the following functions for use with GeoHashes:

- [ST_GeoHash](#) - Returns a GeoHash in the shape of the specified geometry.
- [ST_GeomFromGeoHash](#)- Returns a polygon in the shape of the specified GeoHash.

- [ST_PointFromGeoHash](#) - Returns the center point of the specified GeoHash.

More Details

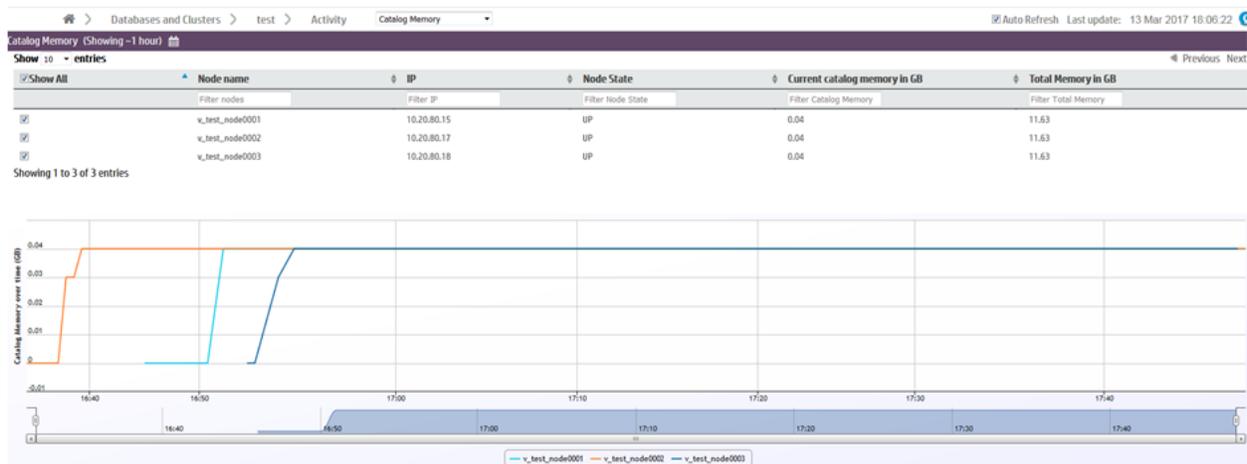
For more information see [Geospatial Analytics](#).

Management Console

This section contains information on updates to the Management Console for Vertica Analytics Platform 8.1.x.

Catalog Memory Activity Page

The new Catalog Memory activity page displays the catalog memory for each node in your database. Use this page to check for sudden changes in catalog memory, or discrepancies in memory distribution across your nodes.



For more about monitoring catalog memory in MC, see [Monitoring Catalog Memory](#).

Configure JVM Settings in MC

Management Console (MC) introduces the ability to configure application server JVM settings on the MC Settings page.

Enabling Extended Monitoring in MC allows you to view more historical monitoring data. For MC to handle larger amounts of monitoring data, Micro Focus recommends increasing JVM initial heap size and maximum heap size. You can now do so on the Configuration tab in MC settings.

Configure WLA Settings in MC

If queries perform sub-optimally, you can get tuning recommendations for them, as well as for hints about optimizing database objects, by using the Workload Analyzer (WLA).

To optimize when WLA uses resources, you can now set WLA to run at a different time for any or all databases that MC monitors. You can also set MC to never run WLA automatically. This setting is available on the Monitoring tab on the MC Settings page.

New Create Cluster Wizard

Vertica 8.1 improves the look and feel of the Create Cluster wizard in MC.

More Details

For more information see [Using Management Console](#) .

SQL Functions and Statements

This section contains information on updates to SQL Functions and Statements for Vertica Analytics Platform 8.1.x:

- [MERGE Support for Update and Insert Filters](#)
- [Adding New Columns to Existing Projections](#)
- [Function to Verify Kerberos Configuration](#)

MERGE Support for Update and Insert Filters

Each `WHEN MATCHED` and `WHEN NOT MATCHED` clause in a `MERGE` statement can now optionally specify an update filter and insert filter, respectively:

```
WHEN MATCHED AND update-filter THEN UPDATE ...  
WHEN NOT MATCHED AND insert-filter THEN INSERT ...
```

Each filter can specify multiple conditions. For details on using these filters, see [Update and Insert Filters](#) in the Administrator's Guide.

Adding New Columns to Existing Projections

Previously, when you added a new column to a table using `ALTER TABLE ADD COLUMN`, existing projections (non-superprojections) did not include the new column. If you wanted to include the new column in your projection, you needed to drop and recreate it. Vertica has added the option `PROJECTIONS` to `ALTER TABLE ADD COLUMN`. When you add a new column to a table, use the `PROJECTIONS` option to simultaneously add the column to one or more existing projections.

For the statement syntax for adding columns to tables, see [ALTER TABLE](#). For an example of adding a new column to an existing projection, see [Adding Table Columns](#).

Function to Verify Kerberos Configuration

You can use the `KERBEROS_CONFIG_CHECK` metafunction to verify your Kerberos configuration. The function tests each element of Kerberos configuration and reports specific errors if it finds any.

More Details

For more information see the [SQL Reference Manual](#).

SDK Updates

This section contains information on updates to the SDK for Vertica Analytics Platform 8.1.x.

Support for C++ 11

The Vertica SDK now supports writing both fenced and unfenced UDxs in C++ 11.

Improvements to Cooperative Parse and Apportioned Load

You can now write UDL functions that support both cooperative parse and apportioned load. Previously a UDL could support only one. Using the C++ API you can now write chunkers that support apportioned load even if the parsers that use them do not. For more information, see [Load Parallelism](#) in Extending Vertica.

More Details

For more information see [Extending Vertica](#).

Apache Kafka Integration

This section contains information on updates to Kafka-integration information for Vertica Analytics Platform 8.1.x.

Execution Parallelism

You can specify the `executionparallelism` parameter in the `KafkaSource` UDL. By default, Vertica automatically creates a thread for every partition. You can use this parameter to throttle the number of threads used to process any `COPY` statement.

Schema Registry Support

Vertica supports the use of a Confluent schema registry with the KafkaAVROParser. By using a schema registry, you enable the Avro parser to parse and decode messages written by the registry and to retrieve schemas stored in the registry. In addition, a schema registry enables Vertica to process streamed data without sending a copy of the schema with each record.

SSL Support

Vertica supports SSL connections between the scheduler, Vertica and Kafka.

More Details

For more information see [Integrating with Apache Kafka](#) .

Apache Spark Integration

This section contains updates to the Apache Spark integration for Vertica Analytics Platform version 8.1.0.

Support for Spark 2.0 and Scala 2.11

The Vertica Connector for Apache Spark now supports Spark 2.0 and Scala 2.11. The connector's JAR file is compatible with a specific version of Spark and Scala. Micro Focus supplies multiple JAR files for download, each of which is compatible with a specific combination of Spark and Scala versions. See [Getting the Spark Connector](#) in [Integrating with Apache Spark](#) for more information.

SaveMode.Append Behavior Change

Previously, the connector returned an error when moving data from Spark to Vertica if you specified SaveMode.Append and the target table did not already exist in Vertica. In version 8.1.0, Vertica creates the table if it does not exist and loads data into it.

Hadoop Integration

This section contains information on updates to Hadoop-integration information for Vertica Analytics Platform 8.1.x.

Rack Locality

When database nodes are co-located on Hadoop nodes, Vertica can take advantage of the Hadoop rack configuration to execute queries. Moving query execution closer to the data reduces network latency and can improve performance.

Vertica automatically uses database nodes that are co-located with the HDFS nodes that contain the data. This behavior, called node locality, requires no additional configuration. In this release, if no database node has local data, Vertica uses a database node in the same rack as the data. Rack locality must be configured. See [Configuring Rack Locality](#) in *Integrating with Apache Hadoop*.

Rack locality is available for reading ORC and Parquet data on co-located clusters.

Function to Verify Kerberos Configuration

You can use the [KERBEROS_HDFS_CONFIG_CHECK](#) metafunction to verify that Vertica can use Kerberos authentication with HDFS. You can have the function test all Hadoop integrations that it finds, or you can test specific clusters or services. The function tests each element of Kerberos configuration and reports specific errors if it finds any.

More Details

For more information see [Integrating with Apache Hadoop](#).

Documentation Changes

The following changes are effective for Vertica 8.1.0.

Document Additions and Revisions

The following additions and revisions have been made to the Vertica product documentation:

- The section Developing UDSFs and UDTFs in R has been re-organized to align with the other SDKs in Extending Vertica. The content can now be found in [Developing with the R SDK](#). This re-organization also includes the creation of the [R SDK API Documentation](#).

Deprecated and Retired Functionality

This section describes the two phases Micro Focus follows to retire Vertica functionality:

- **Deprecated.** Vertica announces deprecated features and functionality in a major or minor release. Deprecated features remain in the product and are functional. Documentation is included in the published release documentation. Accessing the feature can result in informational messages noting that the feature will be removed in the following major or minor release. Vertica identifies deprecated features in this document.
- **Removed.** Micro Focus removes a feature in the major or minor release immediately following the deprecation announcement. Users can no longer access the functionality. Vertica announces all feature removal in this document. Documentation describing the retired functionality is removed, but remains in previous documentation versions.

Deprecated Functionality in This Release

In version 7.0.x the following Vertica functionality was deprecated:

- The `owner` parameter used by the Machine Learning for Predictive Analytics package
- The `key_columns` parameter used by the Machine Learning for Predictive Analytics package
- `vbr --setupconfig` command

See Also

For a description of how Vertica deprecates features and functionality, see [Deprecated and Retired Functionality](#).

Retired Functionality History

The following functionality has been deprecated or removed in the indicated versions:

Functionality	Component	Deprecated Version	Removed Version
Machine Learning for Predictive Analytics package parameter owner.	Server	8.1.1	
Backup and restore --setupconfig command	Server	8.1.1	
vbr --setupconfig command	Server	8.1	
RENAME_MODEL()	Server	8.1	8.1.1
DELETE_MODEL()	Server	8.1	8.1.1
Ability to create a projection in a schema different from its anchor table	Server	8.0.1	8.1.1
PreExcavatorReplicatedProjection()	Server		8.0.1
SET_RECOVER_BY_TABLE(). Do <i>not</i> disable recovery by table.	Server	8.0.1	
Prejoin projections	Server	8.0	
PROJECTIONS_USED column in system table V_MONITOR.QUERY_PROFILES	Optimizer	7.2.2	7.2.3
Administration Tools option --compat21	Server	7.2.1	
ext3 file system SUSE Linux Enterprise Server (SLES) 11.0 Service Pack 3 (SP3) continues to support the ext3 file system.	Server	7.2	8.0
Backup and restore overwrite configuration parameter	Server	7.2	7.2
Buddy projections with different sort order	Server	7.2	
verticaConfig vbr configuration option	Server	7.1	7.2
JavaClassPathForUDx configuration parameter	Server	7.1	

Functionality	Component	Deprecated Version	Removed Version
ADD_LOCATION()	Server	7.1	
bwlimit	Server	7.1	
EXECUTION_ENGINE_PROFILES counters: file handles, memory allocated, and memory reserved	Server	7.0	
MERGE_PARTITIONS()	Server	7.0	
Version 6.0 vbr configuration mapping	Server	7.0	7.2
Administration Tools option check_spread	Server, clients	7.0	
krb5 client authentication method Use the Kerberos gss method for client authentication, instead of krb5. See Configuring Kerberos Authentication .	All clients	7.0	
Pload library	Server	7.0	8.1.1
range-segmentation-clause	Server	6.1.1	
scope parameter of CLEAR_PROFILING()	Server	6.1	
IMPLEMENT_TEMP_DESIGN()	Server, clients	6.1	

Vertica Concepts

This guide introduces the basic concepts to get you started in effectively designing, building, operating, and maintaining a Vertica database. This document assumes that you are familiar with the basic concepts and terminology of relational database management systems and SQL.

The Vertica Approach

The Vertica Analytics Platform consists of the following key features:

Columnar Storage and Execution - column stores offer significant gains in performance, I/O, storage footprint, and efficiency when it comes to analytic workloads. With columnar storage the query only reads the columns needed to answer the query.

Real-time loading and querying - with high query concurrency and the ability to simultaneously load new data into the system, Vertica can load data up to 10X faster than traditional row-store databases.

Advanced Database Analytics - a set of Advanced In-Database Analytics allows you to conduct the analytics computations closer to the data. This provides immediate results from a single place without having to extract data from a separate environment.

Database Designer and Administration Tools - these features allow you to tune and control Vertica with minimal administration effort. For more information see [About Database Designer](#) and [Using the Administration Tools](#).

Advanced Compression - aggressive encoding and compression allows Vertica to dramatically improve analytic performance by reducing CPU, memory, and disk I/O at processing time. Vertica can reduce the original data size to up to 1/10th its original size.

Massively Parallel Processing - a robust and scalable parallel processing solution provides active redundancy, automatic replication, failover, and recovery.

Use Case

A mobile gaming company used to rely on a patchwork of technologies for data warehousing and business intelligence reporting. It took two to four hours to run a query on each game server. The search for a solution led the company to evaluate different companies for expanding its analytic capabilities. The Vertica implementation accomplished the following for the company:

- Queries were reduced from two to four hours to minutes or seconds
- Solution successfully met cloud deployment requirement
- Expanded data capacity from a few months to whole lifetime of data

This has led to better customer support by shortening response time to customer issues, as well as providing the ability to answer many more questions than before the Vertica implementation.

Getting Started

See [Getting Started Overview](#) in the online product documentation and [Get Started With Vertica](#) on our Community site for information on Vertica Analytics Platform implementation.

Vertica Platform

The Vertica Platform provides a unique architecture that allows for efficient processing of custom SQL queries. The architecture also provides a quick return of data. Vertica accomplishes this using the following components:

- Columnar data storage
- Advanced compression
- High Availability
- Automatic Database Design
- Massive Parallel Processing
- Application Integration

Vertica Cluster Architecture

In Vertica, the physical architecture is designed to distribute physical storage and to allow parallel query execution over a potentially large collection of computing resources.

Hybrid Data Store

Vertica stores data on the database in two containers:

- Write Optimized Store (WOS) - stores data in memory without compression or indexing. You can use [INSERT](#), [UPDATE](#), and [COPY](#) to load data into WOS.
- Read Optimized Store (ROS) - stores data on disk. The data is segmented, sorted, and compressed for high optimization. You can load data directly into the ROS using the [COPY](#) statement.

The Tuple Mover moves data from the WOS (memory) to the ROS (disk) using the following processes:

- **Moveout** copies data from the WOS to the Tuple Mover and then to the ROS; data is sorted, encoded, and compressed into column files.
- **Mergeout** combines smaller ROS containers into larger ones to reduce fragmentation.

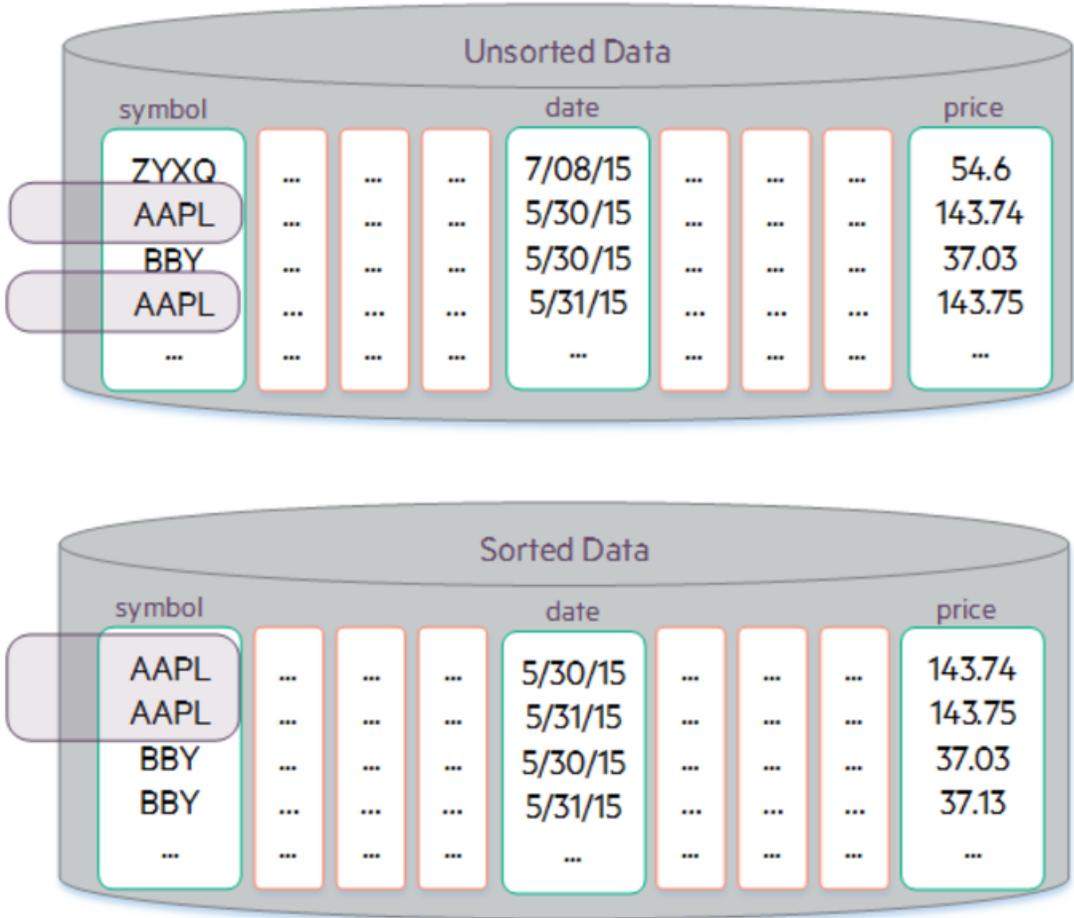
COPY can load data into WOS, where the Tuple Mover moves it to ROS. COPY can also specify loading data directly into the ROS. For more on load options, see [Choosing a Load Method](#).

Column Storage

Vertica stores data in a column format so it can be queried for best performance. Compared to row-based storage, column storage reduces disk I/O making it ideal for read-intensive workloads. Vertica reads only the columns needed to answer the query. For example:

```
=> SELECT avg(price) FROM tickstore WHERE symbol = 'AAPL' and date = '5/31/13';
```

For this example query, a column store reads only three columns while a row store reads all columns:



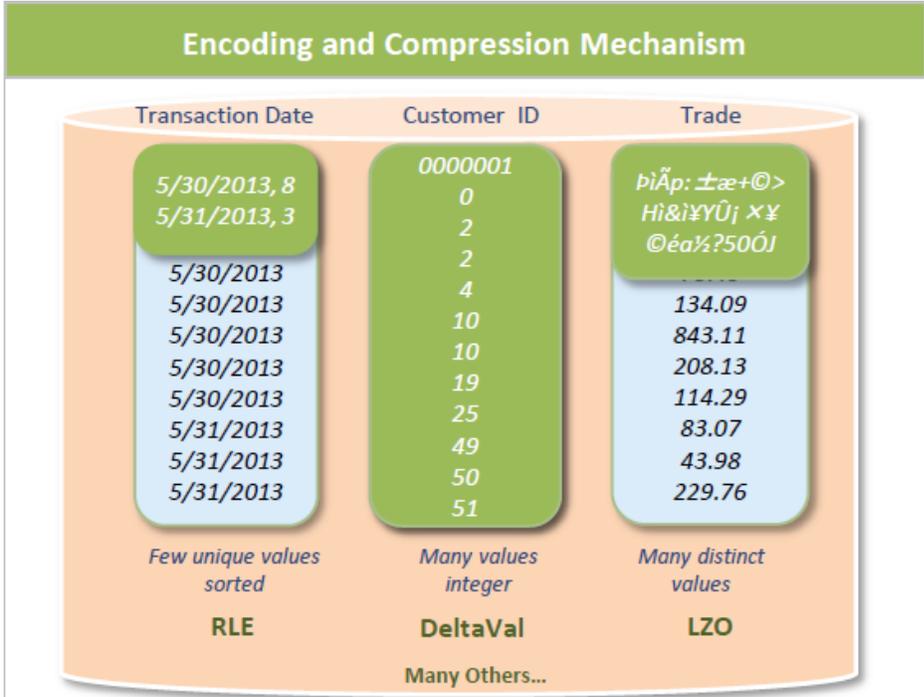
Data Encoding and Compression

Vertica uses encoding and compression to optimize query performance and save storage space.

Encoding converts data into a standard format. Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order. Encoding increases performance because there is less disk I/O during query execution. In addition, you can store more data in less space.

Compression transforms data into a compact format. Vertica uses several different compression methods and automatically chooses the best one for the data being compressed. Using compression, Vertica stores more data, provides more views, and uses less hardware than other databases. Using compression lets you keep much more historical data in physical storage.

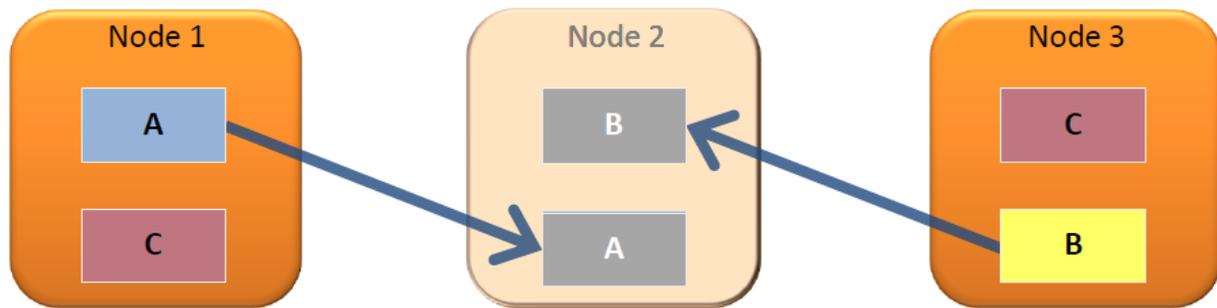
The following shows compression using sorting and cardinality:



For more information, see [Data Encoding and Compression](#).

Clustering

Clustering supports scaling and redundancy. You can scale your database cluster by adding more hardware, and you can improve reliability by distributing and replicating data across your cluster.



Column data gets distributed across nodes in a cluster, so if one node becomes unavailable the database continues to operate. When a node is added to the cluster, or comes back online after being unavailable, it automatically queries other nodes to update its local data.

Projections

A *projection* consists of a set of columns with the same sort order, defined by a column to sort by or a sequence of columns by which to sort. Like an index or materialized view in a traditional database, a projection accelerates query processing. When you write queries in terms of the original tables, the query uses the projections to return query results.

Projections are distributed and replicated across nodes in your cluster, ensuring that if one node becomes unavailable, another copy of the data remains available. For more information, see [K-Safety](#).

Automatic data replication, failover, and recovery provide for *active* redundancy, which increases performance. Nodes recover automatically by querying the system.

Logical and Physical Schema

Vertica stores information about database objects in the logical schema and the physical schema. The difference between the two schemas and how they relate to data storage is an important and unique aspect of the VERTICA architecture.

A *logical* schema consists of objects such as tables, constraints, and views. VERTICA supports any relational schema design that you choose. A *physical* schema consists of collections of table columns called projections. A projection can contain some or all of the columns of a table.

Continuous Performance

Vertica queries and loads data continuously 24x7.

Concurrent loading and querying provides real-time views and eliminates the need for nightly load windows. On-the-fly schema changes allow you to add columns and projections without shutting down your database; Vertica manages updates while keeping the database available.

Terminology

It is helpful to understand the following terms when using Vertica:

Host

A computer system with a 32-bit (non-production use only) or 64-bit Intel or AMD processor, RAM, hard disk, and TCP/IP network interface (IP address and hostname). Hosts share neither disk space nor main memory with each other.

Instance

An instance of Vertica consists of the running Vertica process and disk storage (catalog and data) on a host. Only one instance of Vertica can be running on a host at any time.

Node

A host configured to run an instance of Vertica. It is a member of the database cluster. For a database to have the ability to recover from the failure of a node requires a database K-safety value of at least 1 (3+ nodes).

Cluster

The concept of Cluster in the Vertica Analytics Platform is a collection of hosts with the Vertica software packages (RPM or DEB) that are in one admin tools domain. You can access and manage a cluster from one admintools initiator host.

Database

A cluster of nodes that, when active, can perform distributed data storage and SQL statement execution through administrative, interactive, and programmatic user interfaces.

Note: Although you can define more than one database on a cluster, Vertica supports running only one database per cluster at a time.

Data Encoding and Compression

Vertica uses encoding and compression to optimize query performance and save storage space.

Encoding

Encoding converts data into a standard format and increases performance because there is less disk I/O during query execution. It also passes encoded values to other operations, saving memory bandwidth. Vertica uses several encoding strategies, depending on data type, table cardinality, and sort order. Vertica can directly process encoded data.

Run the Database Designer for optimal encoding in your physical schema. The Database Designer analyzes the data in each column and recommends encoding types for each column in the proposed projections, depending on your design optimization objective. For flex tables, Database Designer recommends the best encoding types for any materialized flex table columns, but not for `__raw__` column projections.

Compression

Compression transforms data into a compact format. Vertica uses integer packing for unencoded integers and LZ0 for compressed data. Before Vertica can process compressed data it must be decompressed.

Compression allows a column store to occupy substantially less storage than a row store. In a column store, every value stored in a column of a projection has the same data type. This greatly facilitates compression, particularly in sorted columns. In a row store, each value of a row can have a different data type, resulting in a much less effective use of compression. Vertica compresses flex table `__raw__` column data by about one half (1/2).

The efficient storage methods that Vertica uses for your database allows you to you maintain more historical data in physical storage.

High Availability

Vertica provides high availability of the database using a RAID-like functionality. This provides the following mechanisms to ensure little to no downtime:

- Multiple copies of the same data on different nodes.
- Vertica continues to load and query data if a node is down.
- Vertica automatically recovers missing data by querying other nodes.

K-Safety

K-safety sets the fault tolerance in the database cluster. The value K represents the number of replicated data in the database cluster. These replicas allow other nodes to take over query processing for any failed nodes.

In Vertica, the value of K can be zero (0), one (1), or two (2). If a database with a K-safety of one (K=1) loses a node, the database continues to run normally. Potentially, the database could continue running if additional nodes fail, as long as at least one other node in the cluster has a copy of the failed node's data. Increasing K-safety to 2 ensures that Vertica can run normally if any two nodes fail. When the failed node or nodes return and successfully recover, they can participate in database operations again.

Note: If the number of failed nodes exceeds the K value, some the data may become unavailable. In this case, the database is considered unsafe and automatically shuts down. However, if every data segment is available on at least one functioning cluster node Vertica continues to run safely.

Potentially, up to half the nodes in a database with a K-safety of 1 could fail without causing the database to shut down. As long as the data on each failed node is available from another active node, the database continues to run.

Note: If half or more of the nodes in the database cluster fail, the database automatically shuts down even if all of the data in the database is available from replicas. This behavior prevents issues due to network partitioning.

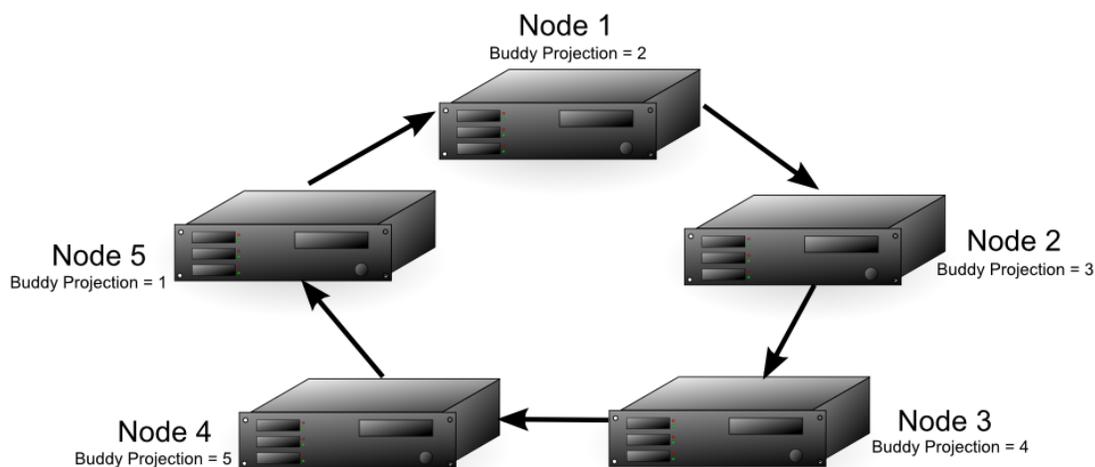
Note: The physical schema design must meet certain requirements. To create designs that are K-safe, Vertica recommends using the Database Designer,.

Buddy Projections

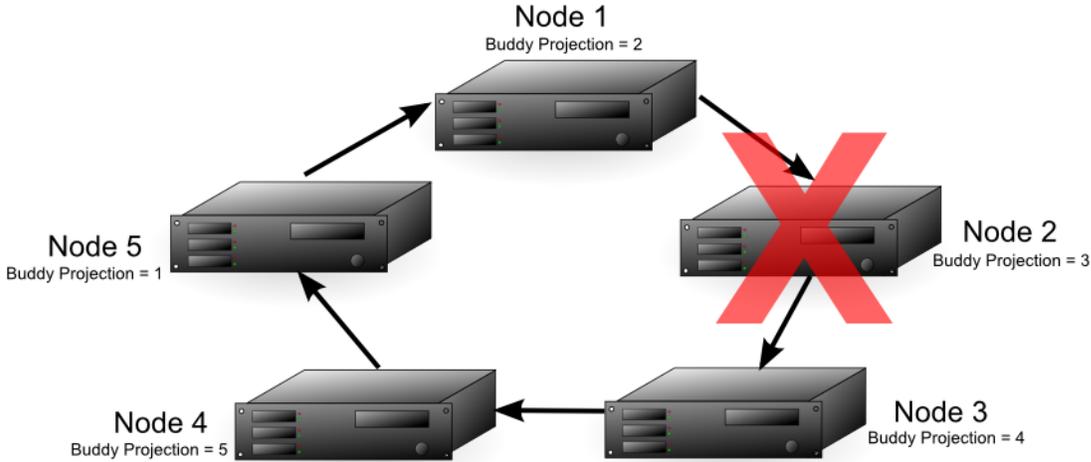
In order to determine the value of k-safety, Vertica creates [buddy projections](#), which are copies of segmented projections distributed across database nodes. (See [Projection Segmentation](#).) Vertica distributes segments that contain the same data to different nodes. This ensures that if a node goes down, all the data is available on the remaining nodes.

K-Safety Example

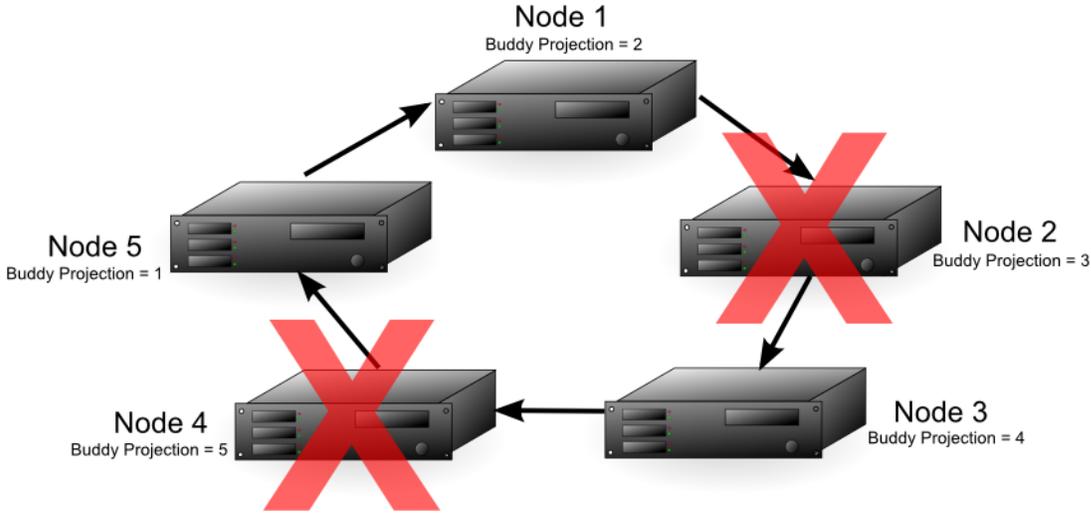
This diagram above shows a 5-node cluster with a K-safety level of 1. Each node contains buddy projections for the data stored in the next higher node (node 1 has buddy projections for node 2, node 2 has buddy projections for node 3, and so on). If any of the nodes fails the database continues to run, though with lower performance, since one of the nodes must handle its own workload and the workload of the failed node.



The diagram below shows a failure of Node 2. In this case, Node 1 handles processing for Node 2 since it contains a replica of node 2's data. Node 1 also continues to perform its own processing. The fault tolerance of the database falls from 1 to 0, since a single node failure could cause the database to become unsafe. In this example, if either Node 1 or Node 3 fails, the database becomes unsafe because not all of its data is available. If Node 1 fails, Node 2's data is no longer available. If Node 3 fails, its data is no longer available, because node 2 is down and could not use the buddy projection. In this case, nodes 1 and 3 are considered critical nodes. In a database with a K-safety level of 1, the node that contains the buddy projection of a failed node, and the node whose buddy projections are on the failed node, always become critical nodes.

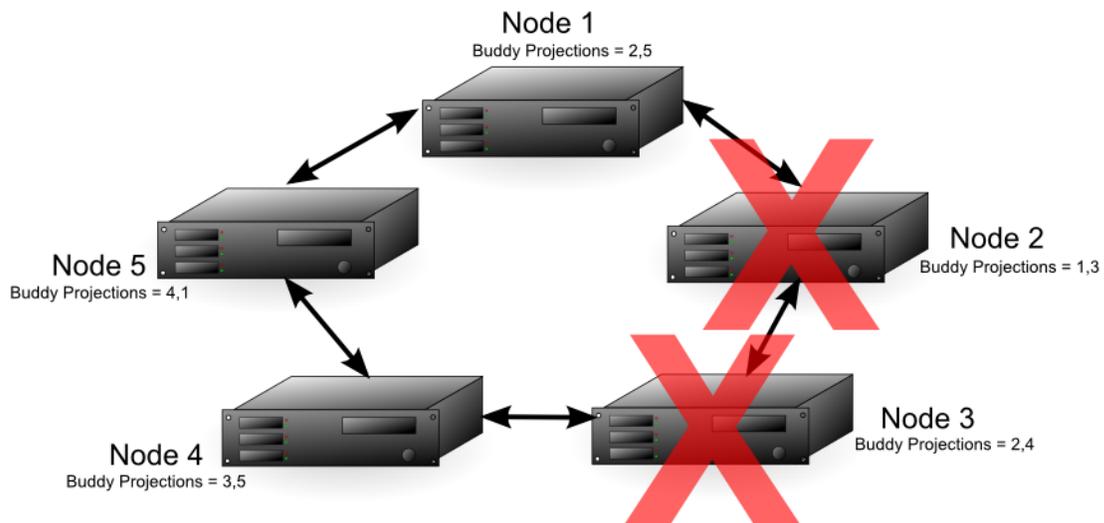


With Node 2 down, either node 4 or 5 could fail and the database still has all of its data available. The diagram below shows that if node 4 fails, node 3 can use its buddy projections to fill in for it. In this case, any further loss of nodes results in a database shutdown, since all the nodes in the cluster are now critical nodes. In addition, if one more node were to fail, half or more of the nodes would be down, requiring Vertica to automatically shut down, no matter if all of the data were available or not.



In a database with a K-safety level of 2, Node 2 and any other node in the cluster could fail and the database continues running. The diagram below shows that each node in the cluster contains buddy projections for both of its neighbors (for example, Node 1 contains buddy projections for Node 5 and Node 2). In this case, nodes 2 and 3 could fail and the database continues running. Node 1 could fill in for Node 2 and Node 4 could fill in for Node 3. Due to the requirement that half or more nodes in the cluster be available in order for the database to

continue running, the cluster could not continue running if node 5 failed, even though nodes 1 and 4 both have buddy projections for its data.



Note: Vertica requires that more than half of all nodes in a cluster must always be available; otherwise, it views the database as being in an unsafe state and shuts it down. Thus, in the previous example, the cluster cannot continue running if Node 5 fails, even though nodes 1 and 4 have buddy projections for its data.

Monitoring K-safety

Use the [SYSTEM](#) table to monitor information related to K-safety, such as:

- `NODE_COUNT`: Number of nodes in the cluster
- `NODE_DOWN_COUNT`: Number of nodes in the cluster that are currently down
- `CURRENT_FAULT_TOLERANCE`: System K-safety level

High Availability With Projections

To ensure high availability and recovery for database clusters of three or more nodes, Vertica:

- Replicates small, unsegmented projections
- Creates buddy projections for large, segmented projections.

Replication (Unsegmented Projections)

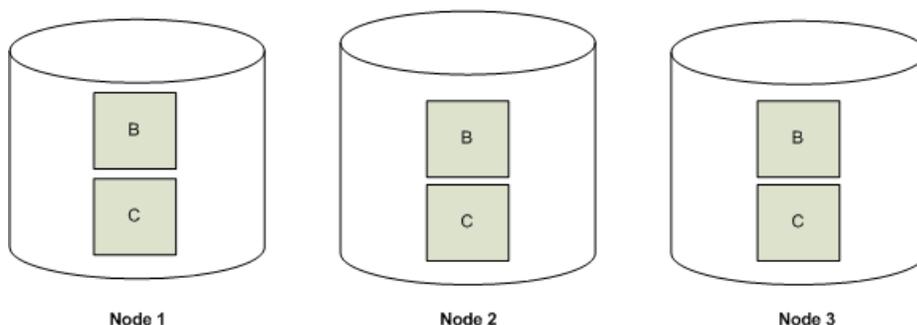
When it creates projections, Database Designer replicates them, creating and storing duplicates of these projections on all nodes in the database.

Replication ensures:

- Distributed query execution across multiple nodes.
- High availability and recovery. In a K-safe database, replicated projections serve as buddy projections. This means that you can use a replicated projection on any node for recovery.

Note: We recommend you use Database Designer to create your physical schema. If you choose not to, be sure to segment all large tables across all database nodes, and replicate small, unsegmented table projections on all database nodes.

The following illustration shows two projections, B and C, replicated across a three node cluster.

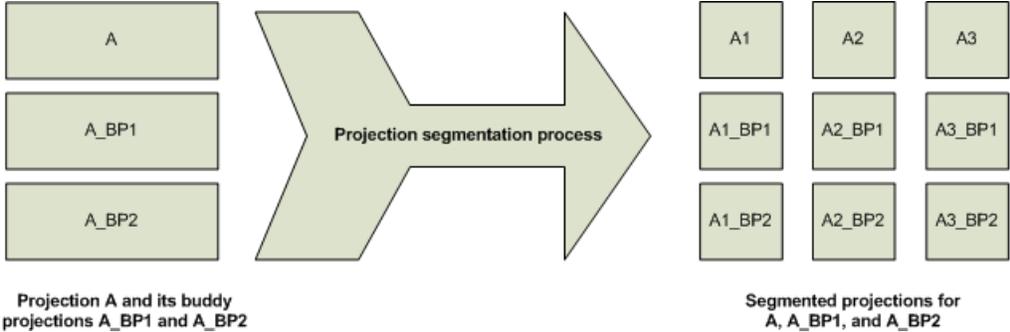


Buddy Projections (Segmented Projections)

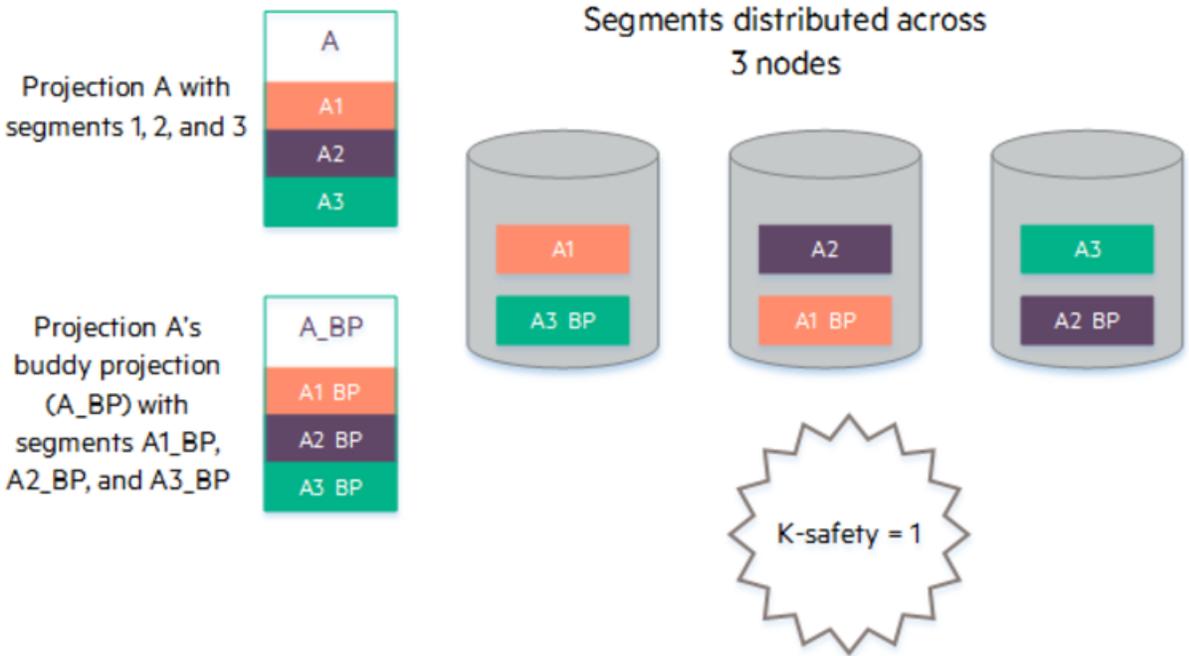
Vertica creates *buddy projections* which are copies of segmented projections that are distributed across database nodes (see [Projection Segmentation](#).) Vertica distributes segments

that contain the same data to different nodes. This ensures that if a node goes down, all the data is available on the remaining nodes. Vertica distributes segments to different nodes by using offsets. For example, segments that comprise the first buddy projection (A_BP1) are offset from projection A by one node, and segments from the second buddy projection (A_BP2) are offset from projection A by two nodes.

The following diagram shows the segmentation for a projection called A and its buddy projections, A_BP1 and A_BP2, for a three node cluster.



The following diagram shows how Vertica uses offsets to ensure that every node has a full set of data for the projection.



How Result Sets Are Stored

Vertica duplicates table columns on all nodes in the cluster to ensure high availability and recovery. Thus, if one node goes down in a K-Safe environment, the database continues to

operate using duplicate data on the remaining nodes. Once the failed node resumes its normal operation, it automatically recovers its lost objects and data by querying other nodes.

Vertica compresses and encodes data to greatly reduce the storage space. It also operates on the encoded data whenever possible to avoid the cost of decoding. This combination of compression and encoding optimizes disk space while maximizing query performance.

Vertica stores table columns as projections. This enables you to optimize the stored data for specific queries and query sets. Vertica provides two methods for storing data:

- Projection segmentation is recommended for large tables (fact and large dimension tables)
- Replication is recommended for the rest of the tables.

High Availability with Fault Groups

Use fault groups to reduce the risk of correlated failures inherent in your physical environment. Correlated failures occur when two or more nodes fail as a result of a single failure. For example, such failures can occur due to problems with shared resources such as power loss, networking issues, or storage.

Vertica minimizes the risk of correlated failures by letting you define fault groups on your cluster. Vertica then uses the fault groups to distribute data segments across the cluster, so the database continues running if a single failure event occurs.

Note: If your cluster layout is managed by a single network switch, a switch failure would cause a single point of failure. Fault groups cannot help with single-point failures.

Vertica supports complex, hierarchical fault groups of different shapes and sizes. You can integrate fault groups with [elastic cluster](#) and [large cluster](#) arrangements to add cluster flexibility and reliability.

Making Vertica Aware of Cluster Topology with Fault Groups

You can also use fault groups to make Vertica aware of the topology of the cluster on which your Vertica database is running. Making Vertica aware of your cluster's topology is required when using Terrace Routing.

Terrace routing is a feature that can reduce the buffer requirements of large queries. Use terrace routing in situations where you have large queries and clusters with a large number of nodes. Without terrace routing, these situations would otherwise require excessive buffer space.

For more information about Terrace Routing, see [Terrace Routing](#).

Automatic Fault Groups

When you configure a cluster of 120 nodes or more, Vertica automatically creates fault groups around control nodes. *Control nodes* are a subset of cluster nodes that manage spread (control messaging). Vertica places nodes that share a control node in the same fault group. See [Large Cluster](#) in the Administrator's Guide for details.

User-Defined Fault Groups

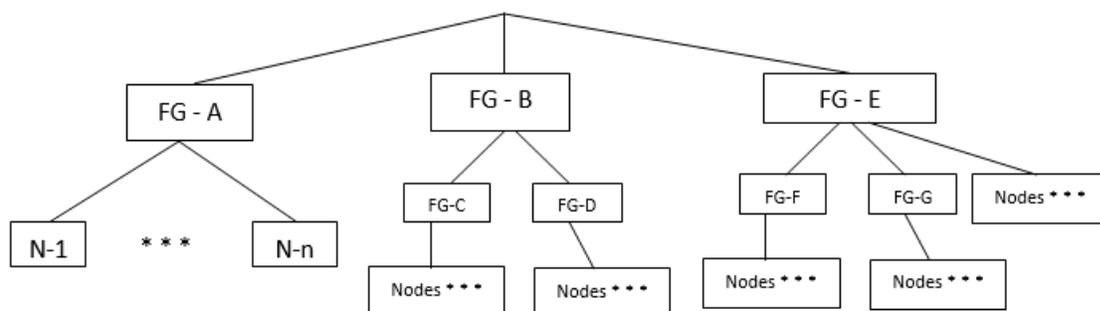
Define your own default groups if:

- Your cluster layout has the potential for correlated failures.
- You want to influence which cluster hosts manage control messaging.

Example Cluster Topology

The following diagram provides an example of hierarchical fault groups configured on a single cluster:

- Fault group FG-A contains nodes only.
- Fault group FG-B (parent) contains child fault groups FG-C and FG-D. Each child fault group also contain nodes.
- Fault group FG-E (parent) contains child fault groups FG-F and FG-G. The parent fault group FG-E also contains nodes.



How to Create Fault Groups

Before you define fault groups, you must have a thorough knowledge of your physical cluster layout. Fault groups require careful planning.

To define fault groups, create an input file of your cluster arrangement. Then, pass the file to a script supplied by Vertica, and the script returns the SQL statements you need to run. See [Fault Groups](#) in the Administrator's Guide for details.

Vertica Components

This section provides an overview of the components that make up Vertica:

- [Logical Schema](#)
- [Physical Schema](#)
- [Database](#)
- [Management Console](#)
- [Administration Tools](#)

These components allow you to tune and control your Vertica Analytics Platform with minimal effort. This eliminates the time and effort a database administrator of a typical database spends identifying issues.

Logical Schema

Design a logical schema for a Vertica database as you would for any SQL database. A logical schema consist of objects such as:

- [Schema](#)
- [Table](#)
- [View](#)
- [Referential Integrity](#)

Vertica supports any relational schema design that you choose.

For more information, see [Designing a Logical Schema](#) in the Administrator's Guide.

Physical Schema

Unlike traditional databases that store data in tables, Vertica physically stores table data in projections, which are collections of table columns.

Projections store data in a format that optimizes query execution. Similar to materialized views, they store result sets on disk rather than compute them each time they are used in a query. Vertica automatically refreshes these result sets with updated or new data.

Projections provide the following benefits:

- Compress and encode data to reduce storage space. Additionally, Vertica operates on the encoded data representation whenever possible to avoid the cost of decoding. This combination of compression and encoding optimizes disk space while maximizing query performance.
- Facilitate distribution across the database cluster. Depending on their size, projections can be segmented or replicated across cluster nodes. For instance, projections for large tables can be segmented and distributed across all nodes. Unsegmented projections for small tables can be replicated across all nodes.
- Transparent to end-users. The Vertica query optimizer automatically picks the best projection to execute a given query.
- Provide high availability and recovery. Vertica duplicates table columns on at least K+1 nodes in the cluster. If one machine fails in a K-Safe environment, the database continues to operate using replicated data on the remaining nodes. When the node resumes normal operation, it automatically queries other nodes to recovers data and lost objects. For more information, see [High Availability with Fault Groups](#) and [High Availability With Projections](#).

Projection Types

A Vertica table typically has multiple projections, each defined to contain different content. Content for the projections of a given table can differ in scope and how it is organized. These differences can generally be divided into the following projection types:

Superprojections

A superprojection contains all the columns of a table. For each table in the database, Vertica requires a minimum of one superprojection.

Under certain conditions, Vertica [automatically creates a table's superprojection](#) immediately on table creation. Vertica also creates a superprojection when you first load data into that table, if none already exists. `CREATE PROJECTION` can create a superprojection if it specifies to include all table columns. A table can have multiple superprojections.

Query-Specific Projections

A query-specific projection is a projection that contains only the subset of table columns to process a given query. Query-specific projections significantly improve the performance of those queries for which they are optimized.

Aggregate Projections

Queries that include expressions or aggregate functions such as [SUM](#) and [COUNT](#) can perform more efficiently when they use projections that already contain the aggregated data. This is especially true for queries on large quantities of data.

Vertica provides three types of projections for storing data that is returned from aggregate functions or expressions:

- [Projection that contains expressions](#): Projection with columns whose values are calculated from anchor table columns.
- [Live aggregate projection](#): Projection that contains columns with values that are aggregated from columns in its anchor table. You can also define live aggregate projections that include [user-defined transform functions](#).
- [Top-K projection](#): Type of live aggregate projection that returns the top k rows from a partition of selected rows. Create a Top-K projection that satisfies the criteria for a Top-K query.

For more information, see [Pre-Aggregating Data in Projections](#).

Projection Segmentation

You can define a projection to maintain its data on the cluster in two ways:

- Divided into multiple segments, or *segmented projections*
- Undivided storage units, or *unsegmented projections*

Segmented Projections

You typically create segmented projections for large fact tables. Vertica splits segmented projections into chunks (segments) of similar size and distributes these segments evenly across the cluster. System K-safety determines how many duplicates (*buddies*) of each segment are created and maintained on different nodes.

You create segmented projections with a `CREATE PROJECTION` statement that includes a `SEGMENTED BY` clause.

Projection segmentation achieves the following goals:

- Ensures high availability and recovery.
- Spreads the query execution workload across multiple nodes.
- Allows each node to be optimized for different query workloads.

Hash Segmentation

Vertica uses hash segmentation to segment large projections. Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution. In a projection, the data to be hashed consists of one or more column values, each having a large number of unique values and an acceptable amount of skew in the value distribution. Primary key columns typically meet these criteria, so they are often used as hash function arguments.

Unsegmented Projections

In many cases, dimension tables are relatively small, so you do not need to segment them. Accordingly, you should design a K-safe database so projections for its dimension tables are replicated without segmentation on all cluster nodes. You create unsegmented projections with a `CREATE PROJECTION` statement that includes the clause `UNSEGMENTED ALL NODES`. This clause specifies to create identical instances of the projection on all cluster nodes.

How Projections are Created

For each table in the database, Vertica requires a minimum of one projection, called a superprojection. A superprojection contains all columns in a given table. Vertica uses superprojections to ensure support for all queries and other DML operations.

Under certain conditions, Vertica [automatically creates a table's superprojection](#) immediately on table creation. Vertica also creates a superprojection when you first load data into that table, if none already exists. [CREATE PROJECTION](#) can create a superprojection if it specifies to include all table columns. A table can have multiple superprojections.

While superprojections can support all queries on any table, they do not facilitate optimal execution of specific queries. To optimize query execution, you should run Database Designer on a representative sample of your data. Database Designer creates projections that optimize your database based on its data statistics and the queries you use, as follows:

1. Analyzes your logical schema, sample data, and (optionally) sample queries.
2. Creates a physical schema, design (projections) in the form of a SQL script that can be deployed automatically or manually.

Database Designer creates designs that provide excellent query performance within physical constraints. Database Designer uses sophisticated strategies to provide excellent ad-hoc query performance while using disk space efficiently. If desired, you can also design [custom projections](#).

Projection Definition Components

[CREATE PROJECTION](#) defines a projection, as in the following example:

```
=> CREATE PROJECTION retail_sales_fact_p (  
    store_key ENCODING RLE,  
    pos_transaction_number ENCODING RLE,  
    sales_dollar_amount,  
    cost_dollar_amount )  
AS SELECT  
    store_key,  
    pos_transaction_number,  
    sales_dollar_amount,  
    cost_dollar_amount  
FROM store.store_sales_fact  
ORDER BY store_key  
SEGMENTED BY HASH(pos_transaction_number) ALL NODES;
```

A projection definition includes the following components:

- [Column List and Encoding](#)
- [Base Query](#)
- [Sort Order](#)
- [Segmentation](#)

Column List and Encoding

This portion of the SQL statement lists every column in the projection and defines the encoding for each column. Vertica supports encoded data, which helps query execution to incur less disk I/O.

```
CREATE PROJECTION retail_sales_fact_P (  
  store_key ENCODING RLE,  
  pos_transaction_number ENCODING RLE,  
  sales_dollar_amount,  
  cost_dollar_amount )
```

Base Query

A projection's base query clause identifies which columns to include in the projection.

```
AS SELECT  
  store_key,  
  pos_transaction_number,  
  sales_dollar_amount,  
  cost_dollar_amount
```

Sort Order

A projection's `ORDER BY` clause determines how to sort projection data. The sort order localizes logically grouped values so a disk read can identify many results at once. For maximum performance, do not sort projections on `LONG VARBINARY` and `LONG VARCHAR` columns. For more information see [ORDER BY Clause](#)

```
ORDER BY store_key
```

Segmentation

A projection's segmentation clause specifies how to distribute projection data across all nodes in the database. Even load distribution helps maximize access to projection data. For large tables, distribute projection data in segments with `SEGMENTED BY HASH`. For example:

```
SEGMENTED BY HASH(pos_transaction_number) ALL NODES;
```

For small tables, use the UNSEGMENTED keyword to replicate table data. Vertica creates identical copies of an unsegmented projection on all cluster nodes. Replication ensures high availability and recovery.

For maximum performance, do not segment projections on LONG VARBINARY and LONG VARCHAR columns.

For more information see [Projection Segmentation](#).

Database

This section covers the following database elements:

- [Database Setup](#)
- [Database Connections](#)
- [Database Security](#)
- [Database Designer](#)
- [Data Loading](#)
- [Workload Management](#)
- [Database Locks](#)

Database Setup

This page provides an overview on setting up a Vertica database . For complete details see [Configuring the Database](#).

Prepare SQL Scripts and Data Files

Prepare the following files before installing Vertica:

- [Logical schema script](#)
- Loadable [data files](#)
- [Load scripts](#)
- [Sample query script](#) (training set)

Create the Database

Create the database after installing Vertica on at least one host:

- Use the Administration Tools to:
 - Create a database
 - Connect to the database
- Use the Database Designer to design the physical schema.
- Use the vsql interactive interface to run SQL scripts that:
 - Create tables and constraints
 - Create projections

Test the Empty Database

- Test for sufficient projections using the sample query script
- Test the projections for K-safety

Test the Partially-Loaded Database

- Load the dimension tables
- Partially load the fact table
- Check system resource usage

- Check query execution times
- Check projection usage

Complete the Fact Table Load

- Monitor system usage
- Complete the fact table load

Set up Security

For security-related tasks, see [Security and Authentication](#) .

- [Optional] Set up SSL
- [Optional] Set up client authentication
- Set up database users and privileges

Set up Incremental Loads

Set up periodic (trickle) loads, see [Trickle Loading](#)

Database Connections

You can connect to a Vertica database in the following ways:

- Interactively using the `vsql` client, as described in [Using vsql](#) in the Administrator's Guide.

`vsql` is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.

You can run `vsql` on any node within a database. To start `vsql`, use the Administration Tools or the shell command described in [Using vsql](#).

- Programmatically using the **JDBC** driver provided by Vertica, as described in [Programming JDBC Client Applications](#) in Connecting to Vertica.

An abbreviation for Java Database Connectivity, JDBC is a call-level application programming interface (API) that provides connectivity between Java programs and data sources (SQL databases and other non-relational data sources, such as spreadsheets or flat files). JDBC is included in the Java 2 Standard and Enterprise editions.

- Programmatically using the **ODBC** driver provided by Vertica, as described in [Programming ODBC Client Applications](#) in Connecting to Vertica.

An abbreviation for Open DataBase Connectivity, ODBC is a standard application programming interface (API) for access to database management systems.

- Programmatically using the **ADO.NET** driver provided by Vertica, as described in [Programming ADO.NET Applications](#) in Connecting to Vertica.

The Vertica driver for ADO.NET allows applications written in C# and Visual Studio to read data from, update, and load data into Vertica databases. It provides a data adapter that facilitates reading data from a database into a data set, and then writing changed data from the data set back to the database. It also provides a data reader ([VerticaDataReader](#)) for reading data and [autocommit](#) functionality for committing transactions automatically.

- Programmatically using **Perl** and the DBI driver, as described in [Programming Perl Client Applications](#) in Connecting to Vertica.

Perl is a free, stable, open source, cross-platform programming language licensed under its Artistic License, or the GNU General Public License (GPL).

- Programmatically using **Python** and the Vertica Python Client or the pyodbc driver, as described in [Programming Python Client Applications](#) in Connecting to Vertica.

Python is a free, agile, object-oriented, cross-platform programming language designed to emphasize rapid development and code readability.

Micro Focus recommends that you deploy Vertica as the only active process on each machine in the cluster and connect to it from applications on different machines. Vertica expects to use all available resources on the machine, and to the extent that other applications are also using these resources, suboptimal performance could result.

Database Security

Vertica secures access to the database and its resources by enabling you to control user access to the database and which tasks users are authorized to perform. See [Security and Authentication](#) .

Database Designer

Vertica's Database Designer is a tool that:

- Analyzes your logical schema, sample data, and, optionally, your sample queries.
- Creates a [Physical Schema](#) design that can be deployed automatically or manually.
- Can be used by anyone without specialized database knowledge. Even business users can run Database Designer.
- Can be run and re-run any time for additional optimization without stopping the database.

Run the DBD

Run the Database Designer in one of the following ways:

- With the Management Console, as described in [Using Management Console to Create a Design](#)
- Programmatically, using the steps described in [About Running Vertica Programmatically](#).
- With the Administration Tools by selecting **Configuration Menu > Run Database Designer**. For details, see [Using the Administration Tools to Create a Design](#)

Use the Database Designer to create one of the following types of designs:

- A [comprehensive design](#) that allows you to create new projections for all tables in your database.
- An [incremental design](#) that creates projections for all tables referenced in the queries you supply.

Database Designer benefits include:

- Accepting up to 100 queries in the query input file for an incremental design.
- Accepting unlimited queries for a comprehensive design.
- Producing higher quality designs by considering UPDATE and DELETE statements.

In most cases, the designs created by Database Designer provide optimal query performance within physical constraints. Database Designer uses sophisticated strategies to provide optimal query performance and data compression.

See Also

- [Physical Schema](#)
- [Creating a Database Design](#)

Data Loading

The SQL Data Manipulation Language (DML) commands INSERT, UPDATE, and DELETE perform the same functions in Vertica as they do in row-oriented databases. These commands follow the SQL-92 transaction model and can be intermixed.

Use the [COPY](#) statement for bulk loading data. COPY reads data from text files or data pipes and inserts it into WOS (memory) or directly into the ROS (disk). COPY can load compressed formats such as GZIP and LZO. COPY automatically commits itself and any current transaction but is not atomic; some rows could be rejected. Note that COPY does not automatically commit when copying data into temporary tables.

You can use the COPY statement's NO COMMIT option to prevent COPY from committing a transaction when it finishes copying data. This allows you to ensure the data in the bulk load is either committed or rolled back at the same time. Also, combining multiple smaller data loads into a single transaction allows Vertica to load the data more efficiently. See the [COPY statement](#) in the SQL Reference Manual for more information.

You can use multiple, simultaneous database connections to load and/or modify data.

For more information about bulk loading, see [Bulk Loading Data](#).

Workload Management

Vertica's resource management scheme allows diverse, concurrent workloads to run efficiently on the database. For basic operations, Vertica pre-configures the built-in [GENERAL pool](#) based on RAM and machine cores. You can customize the General pool to handle specific concurrency requirements.

You can also define new resource pools that you configure to limit memory usage, concurrency, and query priority. You can then optionally assign each database user to use a specific resource pool, which controls memory resources used by their requests.

User-defined pools are useful if you have competing resource requirements across different classes of workloads. Example scenarios include:

- A large batch job takes up all server resources, leaving small jobs that update a web page without enough resources. This can degrade user experience.

In this scenario, create a resource pool to handle web page requests and ensure users get resources they need. Another option is to create a limited resource pool for the batch job, so the job cannot use up all system resources.

- An application has lower priority than other applications and you want to limit the amount of memory and number of concurrent users for the low-priority application.

In this scenario, create a resource pool with an upper limit on the query's memory and associate the pool with users of the low-priority application.

For more information, see [Managing Workload Resources](#) in the Administrator's Guide.

Vertica Database Locks

When multiple users concurrently access the same database information, data manipulation can cause conflicts and threaten data integrity. Conflicts occur because some transactions block other operations until the transaction completes. Because transactions committed at the same time should produce consistent results, Vertica uses locks to maintain data concurrency and consistency. Vertica automatically controls locking by limiting the actions a user can take on an object, depending on the state of that object.

Vertica uses object locks and system locks. *Object locks* are used on objects, such as tables and projections. *System locks* include global catalog locks, local catalog locks, and elastic cluster

locks. Vertica supports a full range of standard SQL [lock modes](#), such as shared (S) and exclusive (X).

For related information about lock usage in different transaction isolation levels, see [READ COMMITTED Isolation](#) and [SERIALIZABLE Isolation](#).

Lock Modes

Vertica has different lock modes that determine how a lock acts on an object. Each lock mode has a lock compatibility and lock strength that reflect how it interacts with other locks in the same environment.

Lock Mode	Description
S - Shared	<p>Use a Shared lock for SELECT queries that run at the serialized transaction isolation level. This allows queries to run concurrently, but the S lock creates the effect that transactions are running in serial order. The S lock ensures that one transaction does not affect another transaction until one transaction completes and its S lock is released.</p> <p>Select operations in READ COMMITTED transaction mode do not require S table locks. See Transactions in Vertica Concepts for more information.</p>
I - Insert	<p>Vertica requires an Insert lock to insert data into a table. Multiple transactions can lock an object in Insert mode simultaneously, enabling multiple inserts and bulk loads to occur at the same time. This behavior is critical for parallel loads and high ingestion rates.</p>
SI - Shared Insert	<p>Vertica requires a Shared Insert lock when both a read and an insert occur in a transaction. Shared Insert mode prohibits delete/update operations. An SI lock also results from lock promotion.</p>
X - Exclusive	<p>Vertica uses Exclusive locks when performing deletes and updates. Only mergeout and moveout operations (U locks) can run concurrently on objects with X locks.</p>
T - Tuple Mover	<p>The Tuple Mover uses T locks for operations on delete vectors. Tuple Mover operations upgrade the table lock mode</p>

	from U to T when work on delete vectors starts so that no other updates or deletes can happen concurrently.
U - Usage	Vertica uses Usage locks for moveout and mergeout Tuple Mover operations. These Tuple Mover operations run automatically in the background, therefore, most other operations (except those requiring an O lock) can run when the object is locked in U mode.
O - Owner	An O lock is the strongest Vertica lock mode. An object acquires an Owner lock when it undergoes changes in both data and structure. Such changes can occur in some DDL operations, such as DROP_PARTITION, TRUNCATE TABLE, and ADD COLUMN. When an object is locked in O mode, it cannot be locked simultaneously by another transaction in any mode.
IV - Insert-Validate	An Insert Validate lock is needed for insert operations where the system performs constraint validation for enabled PRIMARY or UNIQUE key constraints.

Lock Compatibility

Lock compatibility refers to having two locks in effect on the same object at the same time.

Lock Compatibility Matrix

This matrix shows which locks can be used on the same object simultaneously.

When two lock modes intersect in a Yes cell, those modes are compatible. If two requested modes intersect in a No cell, the second request is not granted until the first request releases its lock.

	Granted Mode							
Requested Mode	S	I	IV	SI	X	T	U	O
S	Yes	No	No	No	No	Yes	Yes	No
I	No	Yes	Yes	No	No	Yes	Yes	No

IV	No	Yes	No	No	No	Yes	Yes	No
SI	No	No	No	No	No	Yes	Yes	No
X	No	No	No	No	No	No	Yes	No
T	Yes	Yes	Yes	Yes	No	Yes	Yes	No
U	Yes	No						
O	No	No						

Lock Upgrade Matrix

This matrix shows how your object lock responds to an INSERT request.

If an object has an S lock and you want to do an INSERT, your transaction requests an SI lock. However, if an object has an S lock and you want to perform an operation that requires an S lock, no lock request is issued.

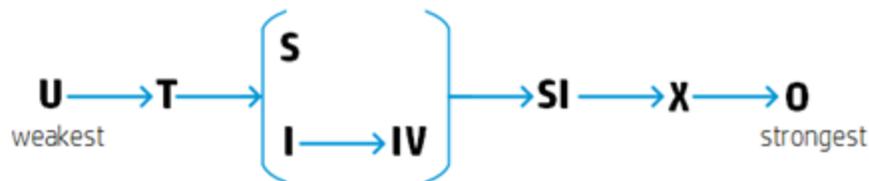
	Granted Mode							
Requested Mode	S	I	IV	SI	X	T	U	O
S	S	SI	SI	SI	X	S	S	O
I	SI	I	IV	SI	X	I	I	O
IV	SI	IV	IV	SI	X	IV	IV	O
SI	SI	SI	SI	SI	X	SI	SI	O
X	X	X	X	X	X	X	X	O
T	S	I	IV	SI	X	T	T	O

U	S	I	IV	SI	X	T	U	O
O	O	O	O	O	O	O	O	O

Lock Strength

Lock strength refers to the ability of a lock mode to interact with another lock mode. O locks are strongest and are incompatible with all other locks. Conversely, U locks are weakest and can run concurrently with all other locks except an O lock.

This figure depicts lock mode strength:



See Also:

- [Vertica Database Locks](#)
- [LOCKS](#)

Troubleshooting Locks

The [LOCKS](#) and [LOCK_USAGE](#) system tables can help identify problems you may encounter with Vertica database locks.

This example shows one row from the LOCKS system table. From this table you can see what types of locks are active on specific objects and nodes.

```

=> SELECT node_names, object_name, lock_mode, lock_scope FROM LOCKS;
node_names      | object_name      | lock_mode | lock_scope
-----+-----+-----+-----
v_vmart_node001 | Table:public.customer_dimension | X          | TRANSACTION
  
```

This example shows two rows from the LOCKS_USAGE system table. You can also use this table to see what locks are in use on specific objects and nodes.

```

=> SELECT node_name, object_name, mode FROM LOCK_USAGE;
node_name      | object_name      | mode
-----+-----+-----
  
```

```
-----+-----+-----  
v_vmart_node0001 | Cluster Topology | S  
v_vmart_node0001 | Global Catalog | X
```

Management Console

Management Console (MC) is a user-friendly performance monitoring and management tool that provides a unified view of your Vertica database operations. Using a browser, you can create, import, manage, and monitor one or more databases and their associated clusters. You can also create and manage MC users. You can then map the MC users to a Vertica database and manage them through the MC interface.

What You Can Do with Management Console

Create...
A database cluster on hosts that do not have Vertica installed
Multiple Vertica databases on one or more clusters from a single point of control
MC users and grant them access to MC and databases managed by MC
Configure...
Database parameters and user settings dynamically
Resource pools
Monitor...
License usage and conformance
Dynamic metrics about your database cluster
Resource pools
User information and activity on MC
Alerts by accessing a single message box of alerts for all managed databases
Recent databases and clusters through a quick link
Import or Export...
Troubleshoot...

Create...
Configure...
Monitor...
Multiple Vertica databases on one or more clusters from a single point of control
Import or Export...
Export all database messages or log/query details to a file
Import multiple Vertica databases on one or more clusters from a single point of control
Troubleshoot...
MC-related issues through a browser

Management Console provides some, but not all, the functionality that Administration Tools provides. Management Console also includes extended functionality not available in admintools. This additional functionality includes a graphical view of your Vertica database and detailed monitoring charts and graphs. See [Administration Tools and Management Console](#) in the Administrator's Guide for more information.

Getting MC

Download the Vertica server RPM and the MC package from [myVertica Portal](#). You then have two options:

- Install Vertica and MC at the command line and import one or more Vertica database clusters into the MC interface
- Install Vertica directly through MC

See the [Installation Guide](#) for details.

What You Need to Know

If you plan to use MC, review the following topics in the Administrator's Guide:

If you want to ...	See ...
Create a new, empty Vertica database	Create a Database on a Cluster
Import an existing Vertica database cluster into MC	Managing Database Clusters
Understand how MC users differ from database users	About MC Users
Read about the MC privilege model	About MC Privileges and Roles
Create new MC users	Creating an MC User
Grant MC users privileges on one or more Vertica databases managed by MC	Granting Database Access to MC Users
Use Vertica functionality through the MC interface	Using Management Console
Monitor MC and Vertica databases managed by MC	Monitoring Vertica Using Management Console
Monitor and configure Resource Pools	Monitoring Resource Pools

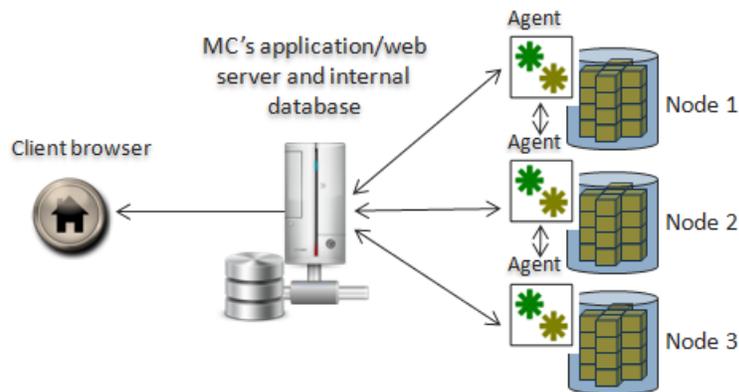
Management Console Architecture

MC accepts HTTP requests from a client web browser, gathers information from the Vertica database cluster, and returns that information to the browser for monitoring.

MC Components

The primary components that drive Management Console are an application/web server and agents that get installed on each node in the Vertica cluster.

The following diagram is a logical representation of MC, the MC user's interface, and the database cluster nodes.



Application/web Server

The application server hosts MC's web application and uses port 5450 for node-to-MC communication and to perform the following:

- Manage one or more Vertica database clusters
- Send rapid updates from MC to the web browser
- Store and report MC metadata, such as alerts and events, current node state, and MC users, on a lightweight, embedded (Derby) database
- Retain workload history

MC Agents

MC agents are internal daemon process that run on each Vertica cluster node. The default agent port, 5444, must be available for MC-to-node and node-to-node communications. Agents monitor MC-managed Vertica database clusters and communicate with MC to provide the following functionality:

- Provide local access, command, and control over database instances on a given node, using functionality similar to Administration Tools.
- Report log-level data from the Administration Tools and Vertica log files.
- Cache details from long-running jobs—such as create/start/stop database operations—that you can view through your browser.
- Track changes to data-collection and monitoring utilities and communicate updates to MC .

- Communicate between all cluster nodes and MC through a webhook subscription, which automates information sharing and reports on cluster-specific issues like node state, alerts, and events.

See Also

- [Monitoring Using MC](#)

Management Console Security

The Management Console (MC) manages multiple Vertica clusters, all which might have different levels and types of security, such as user names and passwords and LDAP authentication. You can also manage MC users who have varying levels of access across these components.

Open Authorization and SSL

Management Console (MC) uses a combination of OAuth (Open Authorization), Secure Socket Layer (SSL), and locally-encrypted passwords to secure HTTPS requests between a user's browser and MC, and between MC and the agents. Authentication occurs through MC and between agents within the cluster. Agents also authenticate and authorize jobs.

The MC configuration process sets up SSL automatically, but you must have the `openssl` package installed on your Linux environment first.

See the following topics in the in the Administrator's Guide for more information:

- [SSL Overview](#)
- [TLS/SSL Server Authentication](#)
- [Generating Certificates and Keys for MC](#)
- [Importing a New Certificate to MC](#)

User Authentication and Access

MC provides two user authentication methods, LDAP or MC. You can use only one method at a time. For example, if you chose LDAP, all MC users will be authenticated against your organization's LDAP server.

You set LDAP authentication up through MC Settings > Authentication on the MC interface.

Note: MC uses LDAP data for authentication purposes only. It does not modify user information in the LDAP repository.

The MC authentication method stores MC user information internally and encrypts passwords. These MC users are not system (Linux) users. They are accounts that have access to MC and, optionally, to one or more MC-managed Vertica databases through the MC interface.

Management Console also has rules for what users can see when they sign in to MC from a client browser. These rules are governed by access levels, each of which is made up of a set of roles.

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [Creating an MC User](#)

Management Console Home Page

The MC Home page is the entry point to all MC-managed Vertica database clusters and MC users. User [access levels](#) determine what a user can see on the MC Home page. Layout and navigation are described in [Using Management Console](#) .

Administration Tools

The Vertica Administration tools allow you to easily perform administrative tasks. You can perform most Vertica database administration tasks with Administration Tools.

Run Administration Tools using the Database Administrator account on the Administration host, if possible. Make sure that no other Administration Tools processes are running.

If the Administration host is unresponsive, run Administration Tools on a different node in the cluster. That node permanently takes over the role of Administration host.

Any user can view the man page available for admintools. Enter the following:

```
man admintools
```

Running Administration Tools

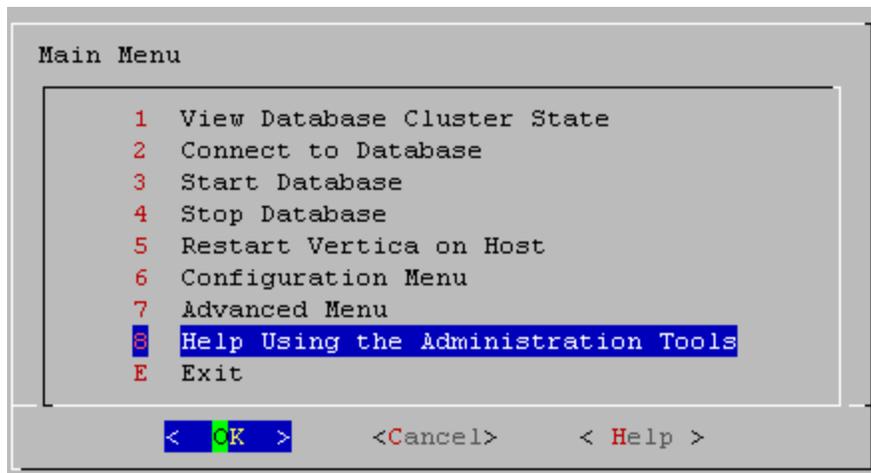
As dbadmin user, you can run administration tools. The syntax follows:

```
/opt/vertica/bin/admintools [--debug ][  
  { -h | --help }  
  | { -a | --help_all }  
  | { -t | --tool } name_of_tool[ options]  
]
```

Options

<code>--debug</code>	If you include the debug option, Vertica logs debug information. Note: You can specify the debug option with or without naming a specific tool. If you specify debug with a specific tool, Vertica logs debug information during tool execution. If you do not specify a tool, Vertica logs debug information when you run tools through the admintools user interface.
<code>-h</code> <code>--help</code>	Outputs abbreviated help.
<code>-a</code> <code>--help_all</code>	Outputs verbose help, which lists all command-line sub-commands and options.
<code>{ -t --tool } <i>name_of_tool</i> [<i>options</i>]</code>	Specifies the tool to run, where <i>name_of_tool</i> is one of the tools described in the help output, and <i>options</i> are one or more comma-delimited tool arguments. Note: Enter <code>admintools -h</code> to see the list of tools available. Enter <code>admintools -t <i>name_of_tool</i> --help</code> to review a specific tool's options.

An unqualified `admintools` command displays the Main Menu dialog box.



If you are unfamiliar with this type of interface, read [Using the Administration Tools Interface](#)

First Login as Database Administrator

The first time you log in as the Database Administrator and run the Administration Tools, the user interface displays.

1. In the end-user license agreement (EULA) window, type `accept` to proceed.

A window displays, requesting the location of the license key file you downloaded from the Micro Focus Web site. The default path is `/tmp/vlicense.dat`.

2. Type the absolute path to your license key (for example, `/tmp/vlicense.dat`) and click **OK**.

Between Dialogs

While the Administration Tools are working, you see the command line processing in a window similar to the one shown below. Do not interrupt the processing.

```
*** Creating database: Stock_Multi ***
Running integrity check
Thread: changing permissions of /opt/vertica/config/share/partinfo.dat: Operation not permitted
Will create database on port 5635
Checking that nodes are defined and installed
stock_multi_node_0 OK [vertica11.2.011288736238288801]
stock_multi_node_1 OK [vertica11.2.011288736238288801]
stock_multi_node_2 OK [vertica11.2.011288736238288801]
Checking full connectivity
Checking/updating multicasting layer
Spread daemon processing
Verifying zspread has been enabled on all nodes
Nominated initiator node stock_multi_node_0
Creating catalog and data directories
Starting DB in multi-node mode
Creating database Stock_Multi
Participating hosts:
  qa0
  qa1
  qa2
processing host qa0
processing host qa1
processing host qa2
Node Status: stock_multi_node_0: (UP)
Creating database nodes
Node Status: stock_multi_node_0: (UP) stock_multi_node_1: (UP) stock_multi_node_2: (UP)
Multi-node start returns: 1
Multi-node DB create completed
Replicating configuration to all nodes
```

SQL in Vertica

Vertica offers a robust set of SQL elements that allow you to manage and analyze massive volumes of data quickly and reliably. Vertica uses the following:

[SQL language elements](#), including:

- Keywords and Reserved Words
- Identifiers
- Literals
- Operators
- Expressions
- Predicates
- Hints

[SQL data types](#), including:

- Binary
- Boolean
- Character
- Date/Time
- Long
- Numeric

[SQL functions](#) including Vertica-specific functions that take advantage of Vertica's unique column-store architecture. For example, call [ANALYZE_STATISTICS](#) to collect and aggregate a variable amount of sample data for statistical analysis.

[SQL statements](#) that let you write robust queries to quickly return large volumes of data.

About Query Execution

When you submit a query, the initiator quickly chooses the projections to use, optimizes and plans the query execution, and logs the SQL statement to its log. This planning results in an Explain Plan. The Explain Plan maps out the steps the query performs. You can view it in the Management Console.

The optimizer breaks down the Explain Plan into smaller plans distributed to [Executor Node](#)

In the final stages of query plan execution, the initiator node does the following:

- Combines results in a grouping operation
- Merges multiple sorted partial result sets from all the executors
- Formats the results to return to the client

For detailed information about writing and executing queries, see [Queries](#) in Analyzing Data.

Backup Isolation Mode

Vertica can run any SQL query in snapshot isolation mode in order to obtain the fastest possible execution. To be precise, snapshot isolation mode is actually a form of a historical query. The syntax is:

```
AT EPOCH LATEST SELECT...
```

The command queries all data in the database up to but not including the current epoch without holding a lock or blocking write operations, which could cause the query to miss rows loaded by other users up to (but no more than) a specific number of minutes before execution.

Historical Queries

Vertica can run a query from a snapshot of the database taken at a specific date and time or at a specific epoch. The syntax is:

```
AT TIME 'timestamp' SELECT...  
AT EPOCH epoch_number SELECT...  
AT EPOCH LATEST SELECT...
```

The command queries all data in the database up to and including the specified epoch or the epoch representing the specified date and time, without holding a lock or blocking write operations. The specified `TIMESTAMP` and `epoch_number` values must be greater than or equal to the Ancient History Mark epoch.

Historical queries are useful because they access data in past epochs only. Historical queries do not need to hold table locks or block write operations because they do not return the absolute latest data. Their content is private to the transaction and valid only for the length of the transaction.

Historical queries behave in the same manner regardless of transaction isolation level. Historical queries observe only committed data, even excluding updates made by the current transaction, unless those updates are to a temporary table.

Be aware that there is only one backup of the logical schema. This means that any changes you make to the schema are reflected across all epochs. If, for example, you add a new column to a table and you specify a default value for the column, all historical epochs display the new column and its default value.

The `DELETE` command in Vertica does not actually delete data; it marks records as deleted. (The `UPDATE` command is actually a combined `INSERT` and a `DELETE`.) Thus, you can control how much deleted data is stored on disk. For more information, see [Managing Disk Space](#) in the Administrator's Guide.

Transactions

When transactions in multiple user sessions concurrently access the same data, session-scoped isolation levels determine what data each transaction can access.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the `SERIALIZABLE` isolation level to ensure consistency.

The Vertica query parser supports standard ANSI SQL-92 isolation levels as follows:

- `READ COMMITTED` (default)
- `READ UNCOMMITTED` : Automatically interpreted as `READ COMMITTED`.
- `REPEATABLE READ`: Automatically interpreted as `SERIALIZABLE`
- `SERIALIZABLE`

Transaction isolation levels `READ COMMITTED` and `SERIALIZABLE` differ as follows:

Isolation level	Dirty read	Non-repeatable read	Phantom read
<code>READ COMMITTED</code>	Not Possible	Possible	Possible
<code>SERIALIZABLE</code>	Not Possible	Not Possible	Not Possible

You can [set separate isolation levels](#) for the database and individual transactions.

Implementation Details

Vertica supports conventional SQL transactions with standard ACID properties:

- ANSI SQL 92 style-implicit transactions. You do not need to run a `BEGIN` or `START TRANSACTION` command.
- No redo/undo log or two-phase commits.
- The `COPY` command automatically commits itself and any current transaction (except when loading temporary tables). It is generally good practice to commit or roll back the current

transaction before you use COPY. This step is optional for DDL statements, which are auto-committed.

Rollback

Transaction rollbacks restore a database to an earlier state by discarding changes made by that transaction. Statement-level rollbacks discard only the changes initiated by the reverted statements. Transaction-level rollbacks discard all changes made by the transaction.

With a ROLLBACK statement, you can explicitly roll back to a named savepoint within the transaction, or discard the entire transaction. Vertica can also initiate automatic rollbacks in two cases:

- An individual statement returns an ERROR message. In this case, Vertica rolls back the statement.
- DDL errors, systemic failures, dead locks, and resource constraints return a ROLLBACK message. In this case, Vertica rolls back the entire transaction.

Explicit and automatic rollbacks always release any locks that the transaction holds.

Savepoints

A *savepoint* is a special marker inside a transaction that allows commands that execute after the savepoint to be rolled back. The transaction is restored to the state that preceded the savepoint.

Vertica supports two types of savepoints:

- An *implicit savepoint* is automatically established after each successful command within a transaction. This savepoint is used to roll back the next statement if it returns an error. A transaction maintains one implicit savepoint, which it rolls forward with each successful command. Implicit savepoints are available to Vertica only and cannot be referenced directly.
- *Named savepoints* are labeled markers within a transaction that you set through [SAVEPOINT](#) statements. A named savepoint can later be referenced in the same transaction through [RELEASE SAVEPOINT](#), which destroys it, and [ROLLBACK TO SAVEPOINT](#), which rolls back all operations that followed the savepoint. Named savepoints can be especially useful

in nested transactions: a nested transaction that begins with a savepoint can be rolled back entirely, if necessary.

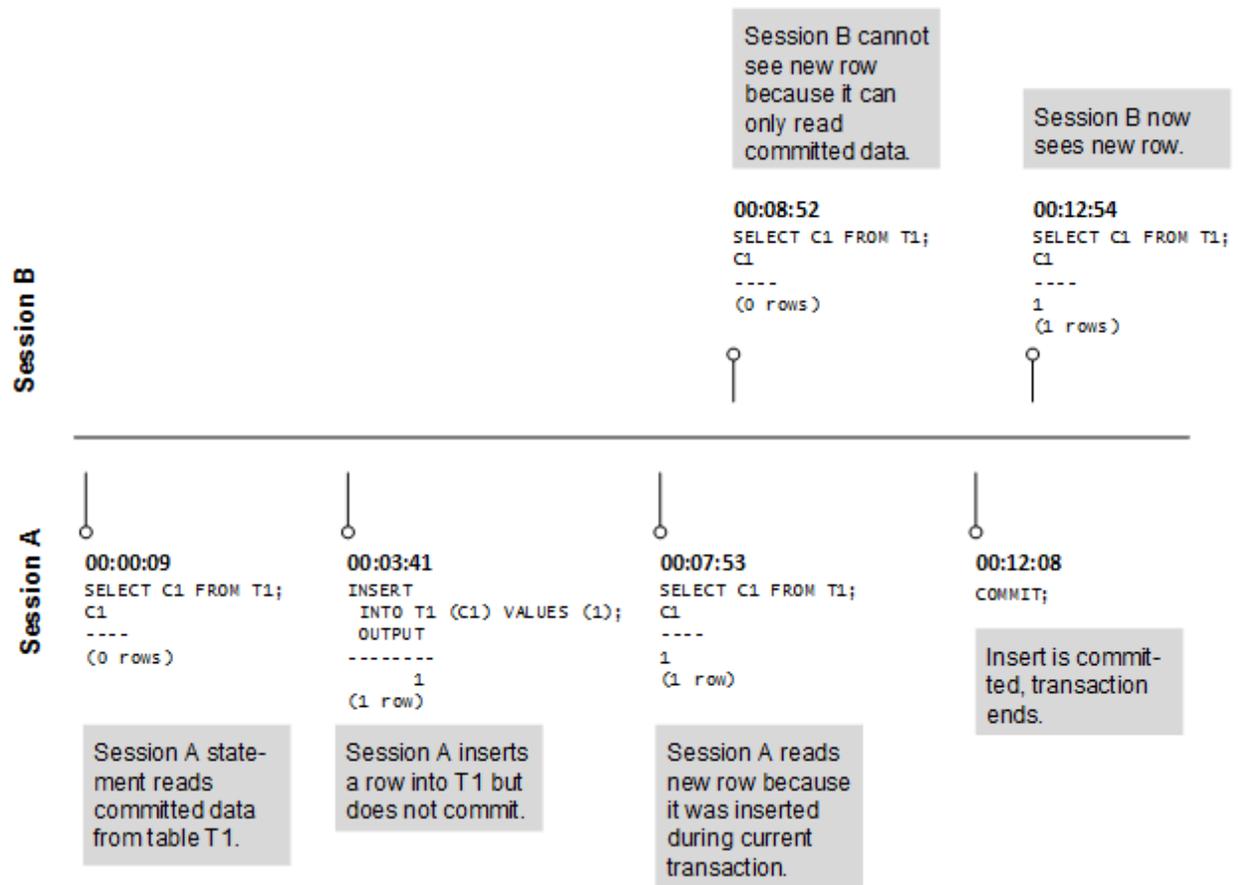
READ COMMITTED Isolation

When you use the isolation level `READ COMMITTED`, a `SELECT` query obtains a backup of committed data at the transaction's start. Subsequent queries during the current transaction also see the results of uncommitted updates that already executed in the same transaction.

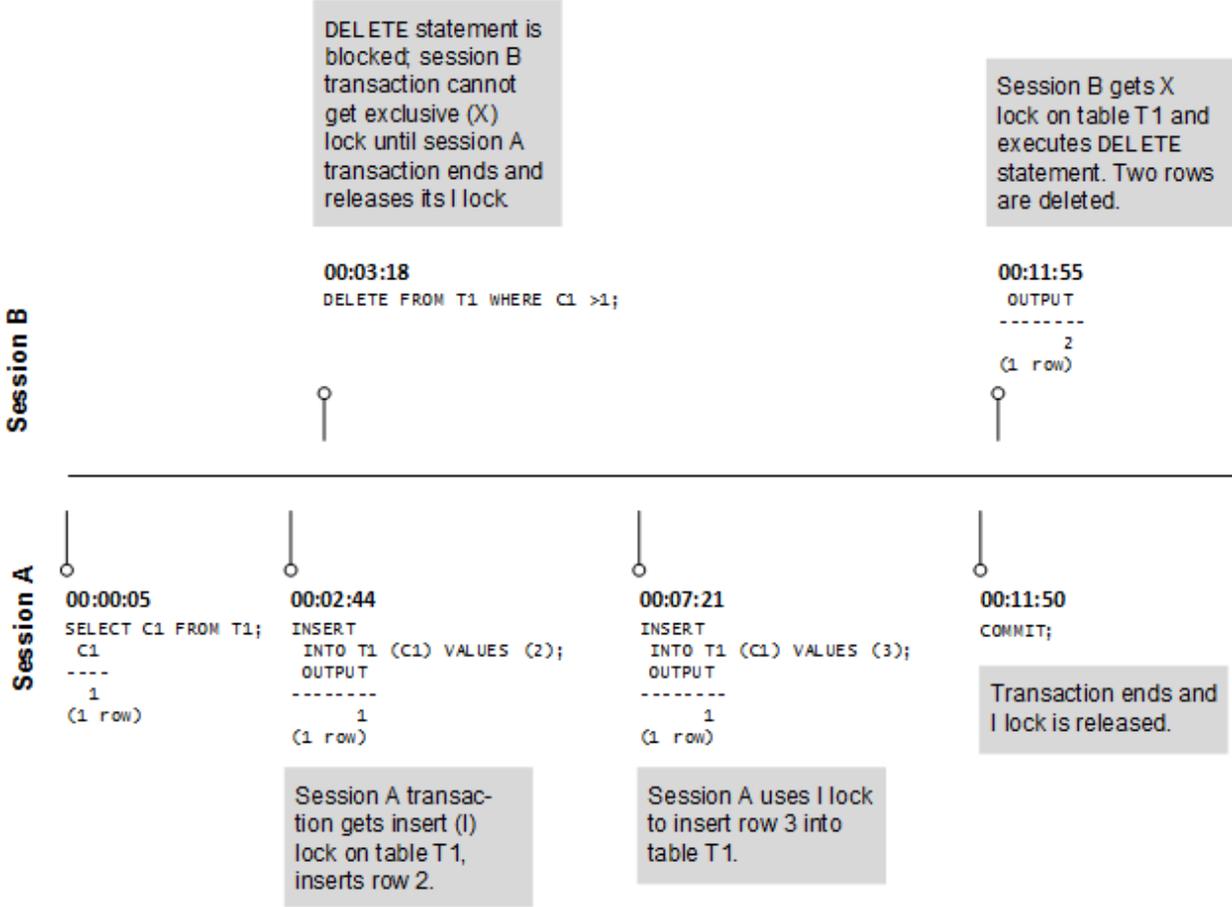
When you use DML statements, your query acquires write locks to prevent other `READ COMMITTED` transactions from modifying the same data. However, be aware that `SELECT` statements do not acquire locks, so concurrent transactions can obtain read and write access to the same selection.

`READ COMMITTED` is the default isolation level. For most queries, this isolation level balances database consistency and concurrency. However, this isolation level can allow one transaction to change the data that another transaction is in the process of accessing. Such changes can yield nonrepeatable and phantom reads. You may have applications with complex queries and updates that require a more consistent view of the database. If so, use [SERIALIZABLE Isolation](#) instead.

The following figure shows how `READ COMMITTED` isolation might control how concurrent transactions read and write the same data:



READ COMMITTED isolation maintains exclusive write locks until a transaction ends, as shown in the following graphic:



See Also

- [Vertica Database Locks](#)
- [LOCKS](#)
- [SET SESSION CHARACTERISTICS AS TRANSACTION](#)
- [Configuration Parameters](#)

SERIALIZABLE Isolation

SERIALIZABLE is the strictest SQL transaction isolation level. While this isolation level permits transactions to run concurrently, it creates the effect that transactions are running in serial order. Transactions acquire locks for read and write operations. Thus, successive SELECT

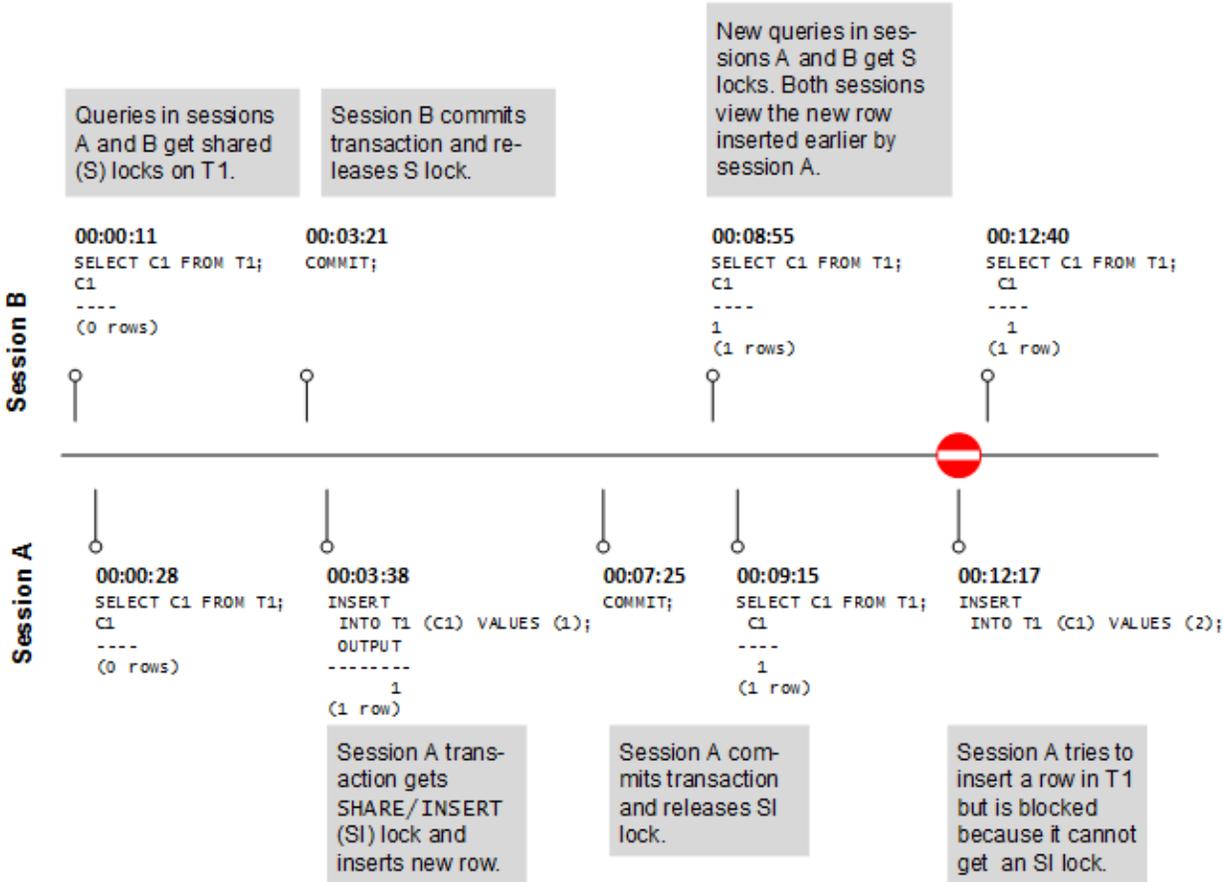
commands within a single transaction always produce the same results. Because SERIALIZABLE isolation provides a consistent view of data, it is useful for applications that require complex queries and updates. However, serializable isolation reduces concurrency. For example, it blocks queries during a bulk load.

SERIALIZABLE isolation establishes the following locks:

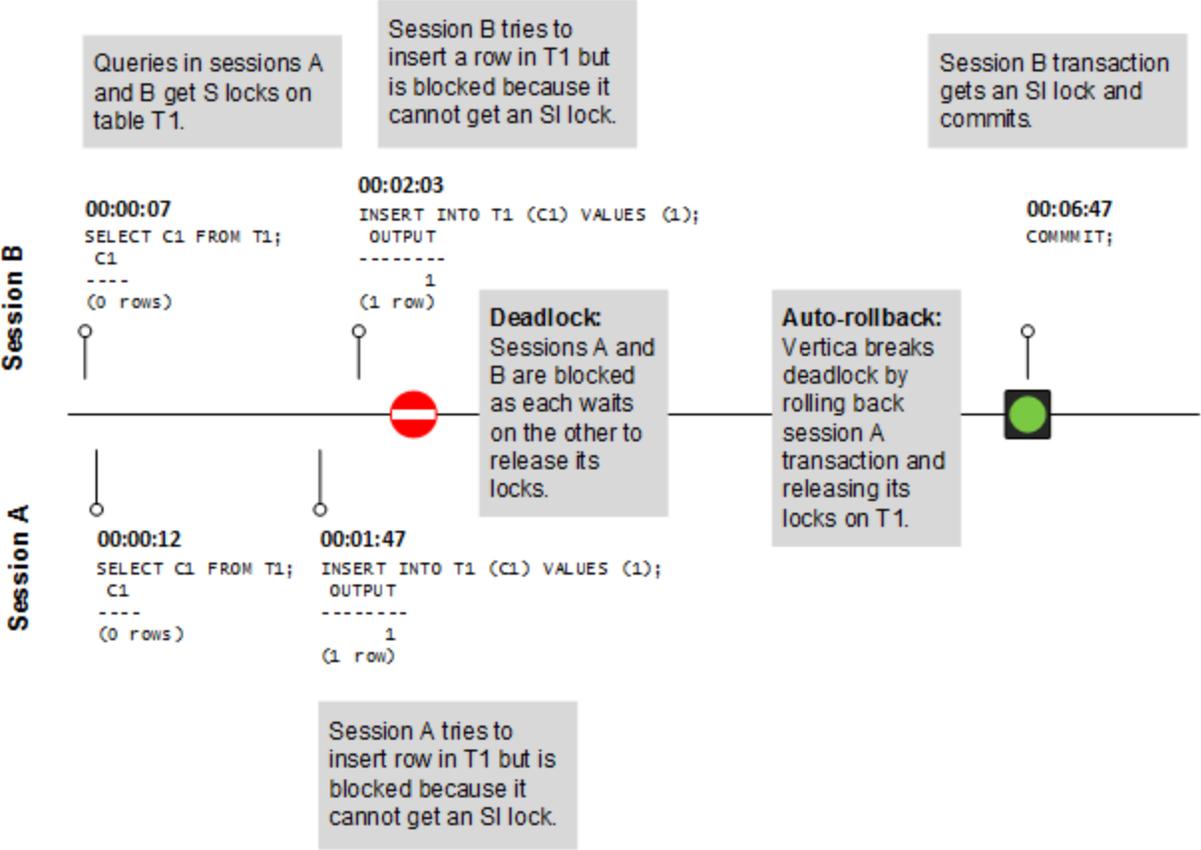
- Table-level read locks: Vertica acquires table-level read locks on selected tables and releases them when the transaction ends. This behavior prevents one transaction from modifying rows while they are being read by another transaction.
- Table-level write lock: Vertica acquires table-level write locks on update and releases them when the transaction ends. This behavior prevents one transaction from reading another transaction's changes to rows before those changes are committed.

At the start of a transaction, a SELECT statement obtains a backup of the selection's committed data. The transaction also sees the results of updates that are run within the transaction before they are committed.

The following figure shows how concurrent transactions that both have SERIALIZABLE isolation levels handle locking:



Applications that use SERIALIZABLE must be prepared to retry transactions due to serialization failures. Such failures often result from deadlocks. When a deadlock occurs, any transaction awaiting a lock automatically times out after 5 minutes. The following figure shows how deadlock might occur and how Vertica handles it:



Note: SERIALIZABLE isolation does not apply to temporary tables. No locks are required for these tables because they are isolated by their transaction scope.

See Also Vertica

- [Vertica Database Locks](#)
- [LOCKS](#)

Extending Vertica

Vertica lets you extend its capabilities to perform new operations or handle new data types, using the following:

- [User-Defined SQL Function](#) allows you to store frequently-used SQL statements.
- [User-Defined Extensions and User-Defined Functions](#) allows you to develop analytic or data-loading tools using programming languages C++, Java, and R.
- [External Procedures](#) allows you to run external scripts installed in your database cluster.

User-Defined SQL Functions

User-defined SQL functions allow you to create and store commonly-used SQL statements. You can use a user-defined SQL function anywhere in a query where an ordinary SQL statement can be used. You need USAGE privileges on the schema and EXECUTE privileges on the function to run a user-defined SQL function.

For information on creating and managing user-defined SQL functions see [Using User-Defined SQL Functions](#).

User-Defined Extensions and User-Defined Functions

User-Defined Extension (UDx) refers to all extensions to Vertica developed using the APIs in the Vertica SDK. UDxs encompass functions such as User-Defined Scalar Functions (UDSFs), and utilities such as the User-Defined Load (UDL) feature that let you create custom data load routines.

Thanks to their tight integration with Vertica, UDxs usually have better performance than User-defined SQL functions or External Procedures.

User-Defined Functions (UDFs) are a specific type of UDx. You use them in SQL statements to process data similarly to Vertica's own built-in functions. They give you the power of creating your own functions that run just slightly slower than Vertica's own function.

The Vertica SDK uses the term UDx extensively, even for APIs that deal exclusively with developing UDFs.

External Procedures

External procedures allow you to call a script or executable program stored in your database cluster. You can pass literal values to this external procedure as arguments. The external procedure cannot communicate back to Vertica.

For information on creating and using external procedures see [Using External Procedures](#).

International Languages and Character Sets

This section describes how Vertica handles internationalization and character sets.

Unicode Character Encoding

Vertica supports Unicode Transformation Format-8, or UTF8, where 8 equals 8-bit. UTF-8 is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard. Initial encoding of byte codes and character assignments for UTF-8 coincides with ASCII. Thus, UTF8 requires little or no change for software that handles ASCII but preserves other values.

Vertica database servers expect to receive all data in UTF-8, and Vertica outputs all data in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. JDBC and ADO.NET APIs operate on data in UTF-16. Client drivers automatically convert data to and from UTF-8 when sending to and receiving data from Vertica using API calls. The drivers do not transform data loaded by executing a [COPY](#) or [COPY LOCAL](#) statement.

See [Implement Locales for International Data Sets](#) in the Administrator's Guide for details.

Locales

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate `GROUP BY` and `ORDER BY` clauses, joins, and the analytic `ORDER BY` clause.

The default locale for a Vertica database is `en_US@collation=binary` (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default `en_US@collation=binary` collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to `en_US_POSIX`. You can set the locale back to the default locale and collation by issuing the `vsq` meta-command `\locale`. For example:

```
=> set locale to '';  
INFO 2567: Canonical locale: 'en_US_POSIX'  
Standard collation: 'LEN'  
English (United States, Computer)  
SET  
=> \locale en_US@collation=binary;  
INFO 2567: Canonical locale: 'en_US'  
Standard collation: 'LEN_KBINARY'  
English (United States)  
=> \locale  
en_US@collation=binary;
```

You can set locale through [ODBC](#), [JDBC](#), and [ADO.net](#).

See the following topics in the Administrator's Guide for details:

- [Implement Locales for International Data Sets](#)
- [Supported Locales](#) in the [Appendix](#)

String Functions

Vertica provides string functions to support internationalization. Unless otherwise specified, these string functions can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of characters. This is accomplished by adding "USING OCTETS" and "USING CHARACTERS" (default) as a parameter to the function.

See [String Functions](#) for details.

Character String Literals

By default, string literals (' . . . ') treat back slashes literally, as specified in the SQL standard.

Tip: If you have used previous releases of Vertica and you do not want string literals to treat back slashes literally (for example, you are using a back slash as part of an escape sequence), you can turn off the `StandardConformingStrings` configuration parameter. See [Internationalization Parameters](#) in the Administrator's Guide. You can also use the `EscapeStringWarning` parameter to locate back slashes which have been incorporated into string literals, in order to remove them.

See [Character String Literals](#) for details.

Installing Vertica

Welcome to Installing Vertica. Read this guide to learn how to prepare for and install the Vertica server. This guide also provides instructions for installing the Vertica Management Console.

For information about installing client drivers, see [Client Drivers](#).

Prerequisites

- This document assumes that you have become familiar with the concepts discussed in Vertica Concepts.
- To perform the procedures described in this document, you must have root password or sudo access (for all commands) for all nodes in your cluster.

Installation Overview and Checklist

This page provides an overview of installation tasks. Carefully review and follow the instructions in all sections in this topic.

Important Notes

- Vertica supports only one running database per cluster.
- Vertica supports installation on one, two, or multiple nodes. The steps for [Installing Vertica](#) are the same, no matter how many nodes are in the cluster.
- Prerequisites listed in [Before You Install Vertica](#) are required for all Vertica configurations.
- Only one instance of Vertica can be running on a host at any time.
- To run the `install_vertica` script, as well as adding, updating, or deleting nodes, you must be logged in as root, or sudo as a user with all privileges. You must run the script for all installations, including upgrades and single-node installations.

Installation Scenarios

The four main scenarios for installing Vertica on hosts are:

- A single node install, where Vertica is installed on a single host as a *localhost* process. This form of install cannot be expanded to more hosts later on and is typically used for development or evaluation purposes.
- Installing to a cluster of physical host hardware. This is the most common scenario when deploying Vertica in a testing or production environment.
- Installing on Amazon Web Services (AWS). When you choose the recommended Amazon Machine Image (AMI), Vertica is installed when you create your instances. For the AWS specific installation procedure, see [Installing and Running Vertica on AWS: The Detailed Procedure](#) rather than the using the steps for installation and upgrade that appear in this guide.

- Installing to a local cluster of virtual host hardware. Also similar to installing on physical hosts, but with network configuration differences.

Before You Install

[Before You Install Vertica](#) describes how to construct a hardware platform and prepare Linux for Vertica installation.

These preliminary steps are broken into two categories:

- Configuring Hardware and Installing Linux
- Configuring the Network

Install or Upgrade Vertica

Once you have completed the steps in the [Before You Install Vertica](#) section, you are ready to run the install script.

[Installing Vertica](#) describes how to:

- Back up any existing databases.
- Download and install the Vertica RPM package.
- Install a cluster using the `install_vertica` script.
- [Optional] [Create a properties file](#) that lets you install Vertica silently.

Note: This guide provides additional [manual procedures](#) in case you encounter installation problems.

- [Upgrading Vertica](#) describes how to upgrade to a more recent version of the software.

Note: If you are upgrading your Vertica license, refer to [Managing Licenses](#) in the Administrator's Guide.

Post-Installation Tasks

[After You Install Vertica](#) describes subsequent steps to take after you've run the installation script. Some of the steps can be skipped based on your needs:

- Install the license key.
- Verify that kernel and user parameters are correctly set.
- Install the vsql client application on non-cluster hosts.
- Resolve any SLES 11.3 issues during spread configuration.
- Use the Vertica documentation online, or download and install Vertica documentation. Find the online documentation and documentation packages to download at <http://my.vertica.com/docs>.
- Install client drivers.
- Extend your installation with Vertica packages.
- [Install](#) or [upgrade](#) the Management Console.

Get started!

- Read the [Concepts Guide](#) for a high-level overview of the Vertica Analytics Platform.
- Proceed to the [Installing and Connecting to the VMart Example Database](#) in Getting Started, where you will be guided through setting up a database, loading sample data, and running sample queries.

About Linux Users Created by Vertica and Their Privileges

This topic describes the Linux accounts that the installer creates and configures so Vertica can run. When you install Vertica, the installation script optionally creates the following Linux user and group:

- dbadmin—Administrative user
- verticadba—Group for DBA users

dbadmin and verticadba are the default names. If you want to change what these Linux accounts are called, you can do so using the installation script. See [Installing Vertica with the Installation Script](#) for details.

Before You Install Vertica

See the following topics for more information:

- [Installation Overview and Checklist](#)
- [General Hardware and OS Requirements and Recommendations](#)

When You Install Vertica

The Linux dbadmin user owns the database catalog and data storage on disk. When you run the install script, Vertica creates this user on each node in the database cluster. It also adds dbadmin to the Linux dbadmin and verticadba groups, and configures the account as follows:

- Configures and authorizes dbadmin for passwordless SSH between all cluster nodes. SSH must be installed and configured to allow passwordless logins. See [Enable Secure Shell \(SSH\) Logins](#).
- Sets the dbadmin user's BASH shell to `/bin/bash`, required to run scripts, such as `install_vertica` and the Administration Tools.
- Provides read-write-execute permissions on the following directories:

- `/opt/vertica/*`
- `/home/dbadmin`—the default directory for database data and catalog files (configurable through the install script)

Note: The Vertica installation script also creates a Vertica database superuser named `dbadmin`. They share the same name, but they are not the same; one is a Linux user and the other is a Vertica user. See [Database Administration User](#) in the Administrator's Guide for information about the database superuser.

After You Install Vertica

Root or sudo privileges are not required to start or run Vertica after the installation process completes.

The `dbadmin` user can log in and perform Vertica tasks, such as creating a database, installing/changing the license key, or installing drivers. If `dbadmin` wants database directories in a location that differs from the default, the root user (or a user with sudo privileges) must create the requested directories and change ownership to the `dbadmin` user.

Vertica prevents administration from users other than the `dbadmin` user (or the user name you specified during the installation process if not `dbadmin`). Only this user can run Administration Tools.

See Also

- [Installation Overview and Checklist](#)
- [Before You Install Vertica](#)
- [Platform Requirements and Recommendations](#)
- [Enable Secure Shell \(SSH\) Logins](#)

Before You Install Vertica

Complete all of the tasks in this section before you install Vertica. When you have completed this section, proceed to [Installing Vertica](#).

Platform Requirements and Recommendations

You must verify that your servers meet the platform requirements described in [Supported Platforms](#). The Supported Platforms topics detail supported versions for the following:

- OS for Server and Management Console (MC)
- Supported Browsers for MC
- Vertica driver compatibility
- R
- Hadoop
- Various plug-ins

BASH Shell

All shell scripts included in Vertica must run under the BASH shell. If you are on a Debian system, then the default shell can be DASH. DASH is not supported. Change the shell for root and for the dbadmin user to BASH with the chsh command.

For example:

```
# getent passwd | grep root
root:x:0:0:root:/root:/bin/dash

# chsh
Changing shell for root.
New shell [/bin/dash]: /bin/bash
Shell changed.
```

Then, as root, change the symbolic link for `/bin/sh` from `/bin/dash` to `/bin/bash`:

```
# rm /bin/sh
# ln -s /bin/bash /bin/sh
```

Log out and back in for the change to take effect.

Install the Latest Vendor Specific System Software

Install the latest vendor drivers for your hardware. For Micro Focus Servers, update to the latest versions for:

- HP ProLiant Smart Array Controller Driver (cciss)
- Smart Array Controller Firmware
- HP Array Configuration Utility (HP ACU CLI)

Data Storage Recommendations

- All internal drives connect to a single RAID controller.
- The RAID array should form one hardware RAID device as a contiguous /data volume.

Validation Utilities

Vertica provides several validation utilities that validate the performance on prospective hosts. The utilities are installed when you install the Vertica RPM, but you can use them before you run the `install_vertica` script. See [Validation Scripts](#) for more details on running the utilities and verifying that your hosts meet the recommended requirements.

General Hardware and OS Requirements and Recommendations

Hardware Recommendations

The Vertica Analytics Platform is based on a massively parallel processing (MPP), shared-nothing architecture, in which the query processing workload is divided among all nodes of the Vertica database. Micro Focus highly recommends using a homogeneous hardware configuration for your Vertica cluster; that is, each node of the cluster should be similar in CPU, clock speed, number of cores, memory, and operating system version.

Note that Micro Focus has not tested Vertica on clusters made up of nodes with disparate hardware specifications. While it is expected that a Vertica database would functionally work in a mixed hardware configuration, performance will most certainly be limited to that of the slowest node in the cluster.

Detailed hardware recommendations are available in [Recommendations for Sizing Vertica Nodes and Clusters](#) (formerly the Vertica Hardware Planning Guide).

Platform OS Requirements

Important: Deploy Vertica as the only active process on each host—other than Linux processes or software explicitly approved by Vertica. Vertica cannot be colocated with other software. Remove or disable all non-essential applications from cluster hosts.

You must verify that your servers meet the platform requirements described in [Vertica Server and Vertica Management Console](#).

Verify Sudo

Vertica uses the sudo command during installation and some administrative tasks. Ensure that sudo is available on all hosts with the following command:

```
# which sudo
/usr/bin/sudo
```

If sudo is not installed, browse to the [Sudo Main Page](#) and install sudo on all hosts.

When you use sudo to install Vertica, the user that performs the installation must have privileges on all nodes in the cluster.

Configuring sudo with privileges for the individual commands can be a tedious and error-prone process; thus, the Vertica documentation does not include every possible sudo command that you can include in the sudoers file. Instead, Micro Focus recommends that you temporarily elevate the sudo user to have all privileges for the duration of the install.

Note: See the sudoers and visudo man pages for the details on how to write/modify a sudoers file.

To allow root sudo access on all commands as any user on any machine, use visudo as root to edit the /etc/sudoers file and add this line:

```
## Allow root to run any commands anywhere
root    ALL=(ALL) ALL
```

After the installation completes, remove (or reset) sudo privileges to the pre-installation settings.

Prepare Disk Storage Locations

You must create and specify directories in which to store your catalog and data files (physical schema). You can specify these locations when you install or configure the database, or later during database operations. Both the catalog and data directories must be owned by the database administrator.

The directory you specify for database catalog files (the catalog path) is used across all nodes in the cluster. For example, if you specify `/home/catalog` as the catalog directory, Vertica uses that catalog path on all nodes. The catalog directory should always be separate from any data file directories.

Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

The data path you designate is also used across all nodes in the cluster. Specifying that data should be stored in `/home/data`, Vertica uses this path on all database nodes.

Do not use a single directory to contain both catalog and data files. You can store the catalog and data directories on different drives, which can be either on drives local to the host (recommended for the catalog directory) or on a shared storage location, such as an external disk enclosure or a SAN.

Before you specify a catalog or data path, be sure the parent directory exists on all nodes of your database. Creating a database in `admintools` also creates the catalog and data directories, but the parent directory must exist on each node.

You do not need to specify a disk storage location during installation. However, you can do so by using the `--data-dir` parameter to the `install_vertica` script. See [Specifying Disk Storage Location During Installation](#)

See Also

- [Specifying Disk Storage Location on MC](#)
- [Specifying Disk Storage Location During Database Creation](#)
- [Configuring Disk Usage to Optimize Performance](#)
- [Using Shared Storage With Vertica](#)

Disk Space Requirements for Vertica

In addition to actual data stored in the database, Vertica requires disk space for several data reorganization operations, such as mergeout and [managing nodes](#) in the cluster. For best results, Micro Focus recommends that disk utilization per node be no more than sixty percent (60%) for a K-Safe=1 database to allow such operations to proceed.

In addition, disk space is temporarily required by certain query execution operators, such as hash joins and sorts, in the case when they cannot be completed in memory (RAM). Such operators might be encountered during queries, recovery, refreshing projections, and so on. The amount of disk space needed (known as temp space) depends on the nature of the queries, amount of data on the node and number of concurrent users on the system. By default, any unused disk space on the data disk can be used as temp space. However, Micro Focus recommends provisioning temp space separate from data disk space. See [Configuring Disk Usage to Optimize Performance](#).

Configuring the Network

This group of steps involve configuring the network. These steps differ depending on your installation scenario. A single node installation requires little network configuration, since the single instance of the Vertica server does not need to communication with other nodes in a cluster. For cluster and cloud install scenarios, you must make several decisions regarding your configuration.

Vertica supports server configuration with multiple network interfaces. For example, you might want to use one as a private network interface for internal communication among cluster hosts (the ones supplied via the `--hosts` option to `install_vertica`) and a separate one for client connections.

Important: Vertica performs best when all nodes are on the same subnet and have the same broadcast address for one or more interfaces. A cluster that has nodes on more than one subnet can experience lower performance due to the network latency associated with a multi-subnet system at high network utilization levels.

Important Notes

- Network configuration is exactly the same for single nodes as for multi-node clusters, with one special exception. If you install Vertica on a single host machine that is to remain a permanent single-node configuration (such as for development or Proof of Concept), you can install Vertica using `localhost` or the loopback IP (typically `127.0.0.1`) as the value for `--hosts`. Do not use the hostname `localhost` in a node definition if you are likely to add nodes to the configuration later.
- If you are using a host with multiple network interfaces, configure Vertica to use the address which is assigned to the NIC that is connected to the other cluster hosts.
- Use a dedicated gigabit switch. If you do not performance could be severely affected.
- Do not use DHCP dynamically-assigned IP addresses for the private network. Use only static addresses or permanently-leased DHCP addresses.

Optionally Run Spread on Separate Control Network

If your query workloads are network intensive, you can use the `--control-network` parameter with the `install_vertica` script (see [Installing Vertica with the Installation Script](#)) to allow spread communications to be configured on a subnet that is different from other Vertica data communications.

The `--control-network` parameter accepts either the default value or a broadcast network IP address (for example, `192.168.10.255`).

Configure SSH

- Verify that root can use Secure Shell (SSH) to log in (`ssh`) to all hosts that are included in the cluster. SSH (SSH client) is a program for logging into a remote machine and for running commands on a remote machine.
- If you do not already have SSH installed on all hosts, log in as root on each host and install it before installing Vertica. You can download a free version of the SSH connectivity tools from [OpenSSH](#).
- Make sure that `/dev/pts` is mounted. Installing Vertica on a host that is missing the mount point `/dev/pts` could result in the following error when you create a database:

```
TIMEOUT ERROR: Could not login with SSH. Here is what SSH said:Last login: Sat Dec 15 18:05:35 2007  
from v_vmart_node0001
```

Allow Passwordless SSH Access for the Dbadmin User

The `dbadmin` user must be authorized for passwordless `ssh`. In typical installs, you won't need to change anything; however, if you set up your system to disallow passwordless login, you'll need to enable it for the `dbadmin` user. See [Enable Secure Shell \(SSH\) Logins](#).

Ensure Ports Are Available

Verify that ports required by Vertica are not in use by running the following command as the root user and comparing it with the ports required in **Firewall Considerations** below:

```
netstat -atupn
```

If you are using a Red Hat 7/CentOS 7 system, use the following command instead:

```
ss -atupn
```

Firewall Considerations

Vertica requires several ports to be open on the local network. Vertica does not recommend placing a firewall between nodes (all nodes should be behind a firewall), but if you must use a firewall between nodes, ensure the following ports are available:

Port	Protocol	Service	Notes
7	TCP	Management Console	Required by Management Console to discover Vertica nodes.
22	TCP	sshd	Required by Administration Tools and the Management Console Cluster Installation wizard.
5433	TCP	Vertica	Vertica client (vsq, ODBC, JDBC, etc) port.
5434	TCP	Vertica	Intra- and inter-cluster communication. Vertica opens the Vertica client port +1 (5434 by default) for intra-cluster communication, such as during a plan. If the port +1 from the default client port is not available, then Vertica opens a random port for intra-cluster communication.
5433	UDP	Vertica	Vertica spread monitoring.
5444	TCP	Vertica Management Console	MC-to-node and node-to-node (agent) communications port. See Changing MC or Agent Ports .
5450	TCP	Vertica Management Console	Port used to connect to MC from a web browser and allows

Port	Protocol	Service	Notes
			communication from nodes to the MC application/web server. See Connecting to Management Console .
4803	TCP	Spread	Client connections.
4803	UDP	Spread	Daemon to Daemon connections.
4804	UDP	Spread	Daemon to Daemon connections.
6543	UDP	Spread	Monitor to Daemon connection.

Operating System Configuration Task Overview

This topic provides a high-level overview of the OS settings required for Vertica. Each item provides a link to additional details about the setting and detailed steps on making the configuration change. The installer tests for all of these settings and provides hints, warnings, and failures if the current configuration does not meet Vertica requirements.

Before You Install the Operating System

Configuration	Description
Supported Platforms	Verify that your servers meet the platform requirements described in Supported Platforms . Unsupported operating systems are detected by the installer.
LVM	Vertica Analytics Platform supports Linux Volume Manager (LVM) on all supported operating systems. For information on LVM requirements and restrictions, see the section, Vertica Support for LVM .
Filesystem	The filesystem for the Vertica data and catalog directories must be formatted as ext4.
Swap Space	A 2GB swap partition is required. Partition the remaining disk space in a single partition under "/".
Disk Block Size	The disk block size for the Vertica data and catalog directories should be 4096 bytes, the default for ext4 filesystems.
Memory	For more information on sizing your hardware, see the Vertica Hardware Planning Guide .

Firewall Considerations

Configuration	Description
Firewall/Ports	Firewalls, if present, must be configured so as not to interfere with Vertica.

General Operating System Configuration - Automatically Configured by Installer

These general OS settings are automatically made by the installer if they do not meet Vertica requirements. You can prevent the installer from automatically making these configuration changes by using the `--no-system-configuration` parameter for the `install_vertica` script.

Configuration	Description
Nice Limits	The database administration user must be able to <i>nice</i> processes back to the default level of 0.
min_free_kbytes	The <code>vm.min_free_kbytes</code> setting in <code>/etc/sysctl.conf</code> must be configured sufficiently high. The specific value depends on your hardware configuration.
User Open Files Limit	The open file limit for the <code>dbadmin</code> user should be at least 1 file open per MB of RAM, 65536, or the amount of RAM in MB; whichever is greater.
System Open File Limits	The maximum number of files open on the system must not be less than at least the amount of memory in MB, but not less than 65536.
Pam Limits	<code>/etc/pam.d/su</code> must contain the line: <code>session required pam_limits.so</code> This allows for the conveying of limits to commands run with the <code>su</code> - command.
Address Space Limits	The address space limits (<code>as</code> setting) defined in <code>/etc/security/limits.conf</code> must be unlimited for the database administrator.
File Size Limits	The file size limits (<code>fsize</code> setting) defined in <code>/etc/security/limits.conf</code> must

Configuration	Description
	be unlimited for the database administrator.
User Process Limits	The nproc setting defined in /etc/security/limits.conf must be 1024 or the amount of memory in MB, whichever is greater.
Maximum Memory Maps	The vm.max_map_count in /etc/sysctl.conf must be 65536 or the amount of memory in KB / 16, whichever is greater.

General Operating System Configuration - Manual Configuration

The following general OS settings must be done manually.

Configuration	Description
Disk Readahead	This disk readahead must be at least 2048, with a high of 8192. Set this high limit only with the help of Vertica support. The specific value depends on your hardware configuration.
NTP Services	The NTP daemon must be enabled and running, with the exception of Red Hat 7 and CentOS 7 systems.
chrony	For Red Hat 7 and CentOS 7 systems, chrony must be enabled and running.
SELinux	SELinux must be disabled or run in permissive mode.
CPU Frequency Scaling	Vertica recommends that you disable CPU Frequency Scaling. Important: Your systems may use significantly more energy when CPU frequency scaling is disabled.
Transparent Hugepages	For Red Hat 7 and CentOS 7 systems, Transparent Hugepages must be set to <i>always</i> . For all other operating systems, Transparent Hugepages must be disabled or set to <i>advise</i> .
I/O Scheduler	The I/O Scheduler for disks used by Vertica must be set to <i>deadline</i> or <i>noop</i> .

Configuration	Description
Support Tools	Several optional packages can be installed to assist Vertica support when troubleshooting your system.

System User Requirements

The following tasks pertain to the configuration of the system user required by Vertica.

Configuration	Required Setting(s)
System User Requirements	The installer automatically creates a user with the correct settings. If you specify a user with <code>--dba-user</code> , then the user must conform to the requirements for the Vertica system user.
LANG Environment Settings	The LANG environment variable must be set and valid for the database administration user.
TZ Environment Settings	The TZ environment variable must be set and valid for the database administration user.

Before You Install The Operating System

The topics in this section detail system settings that must be configured when you install the operating system. These settings cannot be easily changed after the operating system is installed.

Supported Platforms

The Vertica installer checks the type of operating system that is installed. If the operating system does not meet one of the supported operating systems (See [Vertica Server and Vertica Management Console](#)), or the operating system cannot be determined, then the installer halts.

The installer generates one of the following issue identifiers if it detects an unsupported operating system:

- [S0320] - Fedora OS is not supported.
- [S0321] - The version of Red Hat/CentOS is not supported.
- [S0322] - The version of Ubuntu/Debian is not supported.
- [S0323] - The operating system could not be determined. The unknown operating system is not supported because it does not match the list of supported operating systems.
- [S0324] - The version of Red Hat is not supported.

Filesystem Requirement

Vertica requires that your Linux filesystem be ext4. All other filesystem types are unsupported. The installer reports this issue with the identifier **S0160**.

Swap Space Requirements

Vertica requires at least 2 GB swap partition regardless of the amount of RAM installed on your system. The installer reports this issue with identifier **S0180**.

For typical installations Vertica recommends that you partition your system with a 2GB primary partition for swap regardless of the amount of installed RAM. Larger swap space is acceptable, but unnecessary.

Note: Do not place a swap file on a disk containing the Vertica data files. If a host has only two disks (boot and data), put the swap file on the boot disk.

If you do not have at least a 2 GB swap partition then you may experience performance issues when running Vertica.

You typically define the swap partition when you install Linux. See your platform's documentation for details on configuring the swap partition.

Disk Block Size Requirements

Vertica recommends that the disk block size be 4096 bytes, the default on ext4 filesystems. The installer reports this issue with the identifier **S0165**.

The disk block size is set when you format your file system. Changing the block size requires a re-format.

Memory Requirements

Vertica requires, at a minimum, 1GB of RAM per logical processor. The installer reports this issue with the identifier **S0190**.

However, for performance reasons, you typically require more RAM than the minimum. For more information on sizing your hardware, see the [Vertica Hardware Planning Guide](#).

Firewall Considerations

Vertica requires multiple ports be open between nodes. You may use a firewall (IP Tables) on Redhat/CentOS and Ubuntu/Debian based systems. Note that firewall use is not supported on SuSE systems and that SuSE systems must disable the firewall. The installer reports issues found with your IP tables configuration with the identifiers **N0010** for (systems that use IP Tables) and **N011** (for SuSE systems).

The installer checks the IP tables configuration and issues a warning if there are any configured rules or chains. The installer does not detect if the configuration may conflict with Vertica. It is your responsibility to verify that your firewall allows traffic for Vertica as described in [Ensure Ports Are Available](#).

Note: The installer does not check NAT entries in iptables.

You can modify your firewall to allow for Vertica network traffic, or you can disable the firewall if your network is secure. Note that firewalls are not supported for Vertica systems running on SuSE.

Important: You may encounter the **N0010** issue even when the firewall is disabled. If this occurs, you can work around this issue and install Vertica by ignoring installer WARN messages. To do this, install (or update) with a failure threshold of FAIL. For example, `/opt/vertica/sbin/install_vertica --failure-threshold FAIL <other install options...>`.

Red Hat 6 and CentOS 6 Systems

For details on how to configure iptables and allow specific ports to be open, see the platform-specific documentation for your platform:

- RedHat: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sect-Security_Guide-IPTables.html
- CentOS: <http://wiki.centos.org/HowTos/Network/IPTables>

To disable iptables, run the following command as root or sudo:

```
# service iptables save
# service iptables stop
# chkconfig iptables off
```

To disable iptables if you are using the ipv6 versions of iptables, run the following command as root or sudo:

```
# service ip6tables save
# service ip6tables stop
# chkconfig ip6tables off
```

Red Hat 7 and CentOS 7 Systems:

To disable the system firewall, run the following command as root or sudo:

```
# systemctl mask firewalld
# systemctl disable firewalld
# systemctl stop firewalld
```

Ubuntu and Debian Systems

For details on how to configure iptables and allow specific ports to be open, see the platform-specific documentation for your platform:

- Debian: <https://wiki.debian.org/iptables>
- Ubuntu: <https://help.ubuntu.com/12.04/serverguide/firewall.html>.

Note: Ubuntu uses the ufw program to manage iptables.

To disable iptables on Debian, run the following command as root or sudo:

```
/etc/init.d/iptables stop

update-rc.d -f iptables remove
```

To disable iptables on Ubuntu, run the following command:

```
sudo ufw disable
```

SuSE Systems

The firewall must be disabled on SUSE systems. To disable the firewall on SuSE systems, run the following command:

```
/sbin/SuSEfirewall12 off
```

Port Availability

The `install_vertica` script checks that required ports are open and available to Vertica. The installer reports any issues with the identifier: **N0020**.

Port Requirements

The following table lists the ports required by Vertica.

Port	Protocol	Service	Notes
7	TCP	Management Console	Required by Management Console to discover Vertica nodes.
22	TCP	sshd	Required by Administration Tools and the Management Console Cluster Installation wizard.
5433	TCP	Vertica	Vertica client (vsq , ODBC, JDBC, etc) port.
5434	TCP	Vertica	Intra- and inter-cluster communication. Vertica opens the Vertica client port +1 (5434 by default) for intra-cluster communication, such as during a plan. If the port +1 from the default client port is not available, then Vertica opens a random port for intra-cluster communication.

Port	Protocol	Service	Notes
5433	UDP	Vertica	Vertica spread monitoring.
5444	TCP	Vertica Management Console	MC-to-node and node-to-node (agent) communications port. See Changing MC or Agent Ports .
5450	TCP	Vertica Management Console	Port used to connect to MC from a web browser and allows communication from nodes to the MC application/web server. See Connecting to Management Console .
4803	TCP	Spread	Client connections.
4803	UDP	Spread	Daemon to Daemon connections.
4804	UDP	Spread	Daemon to Daemon connections.
6543	UDP	Spread	Monitor to Daemon connection.

General Operating System Configuration - Automatically Configured by the Installer

These general Operating System settings are automatically made by the installer if they do not meet Vertica requirements. You can prevent the installer from automatically making these configuration changes by using the `--no-system-configuration` parameter for the `install_vertica` script.

sysctl

During installation, Vertica attempts to automatically change various OS level settings. The installer may not change values on your system if they exceed the threshold required by the installer. You can prevent the installer from automatically making these configuration changes by using the `--no-system-configuration` parameter for the `install_vertica` script.

To permanently edit certain settings and prevent them from reverting on reboot, use `sysctl`.

The `sysctl` settings relevant to the installation of Vertica include:

- [min_free_kbytes](#)
- [fs.file_max](#)
- [vm.max_map_count](#)

Permanently Changing Settings with sysctl:

1. As the root user, open the `/etc/sysctl.conf` file:

```
# vi /etc/sysctl.conf
```

2. Enter a parameter and value:

```
parameter = value
```

For example, to set the parameter and value for `fs.file-max` to meet Vertica requirements, enter:

```
fs.file-max = 65536
```

3. Save your changes, and close the `/etc/sysctl.conf` file.
4. As the root user, reload the config file:

```
# sysctl -p
```

Identifying Settings Added by the Installer

You can see whether the installer has added a setting by opening the `/etc/sysctl.conf` file:

```
# vi /etc/sysctl.conf
```

If the installer has added a setting, the following line appears:

```
# The following 1 line added by Vertica tools. 2015-02-23 13:20:29  
parameter = value
```

Nice Limits Configuration

The Vertica system user (`dbadmin` by default) must be able to raise and lower the priority of Vertica processes. To do this, the `nice` option in the `/etc/security/limits.conf` file must

include an entry for the dbadmin user. The installer reports this issue with the identifier: **S0010**.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

Note: Vertica never raises priority above the default level of 0. However, Vertica does lower the priority of certain Vertica threads and needs to be able to raise the priority of these threads back up to the default level. This setting allows Vertica to raise the priorities back to the default level.

All Systems

To set the Nice Limit configuration for the dbadmin user, edit `/etc/security/limits.conf` and add the following line. Replace *dbadmin* with the name of your system user.

```
dbadmin - nice 0
```

min_free_kbytes Setting

This topic details how to update the `min_free_kbytes` setting so that it is within the range supported by Vertica. The installer reports this issue with the identifier: **S0050** if the setting is too low, or **S0051** if the setting is too high.

The `vm.min_free_kbytes` setting configures the page reclaim thresholds. When this number is increased the system starts reclaiming memory earlier, when its lowered it starts reclaiming memory later. The default `min_free_kbytes` is calculated at boot time based on the number of pages of physical RAM available on the system.

The setting must be the greater of:

- The default value configured by the system, or
- 4096, or
- determine the value from running the command below.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-`

`system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To manually set `min_free_kbytes`:

1. Determine the current/default setting with the following command:

```
/sbin/sysctl vm.min_free_kbytes
```

2. If the result of the previous command is `No such file or directory` or the default value is less than 4096, then run the command below:

```
memtot=`grep MemTotal /proc/meminfo | awk '{printf "%.0f", $2}'`  
echo "scale=0;sqrt ($memtot*16)" | bc
```

3. Edit or add the current value of `vm.min_free_kbytes` in `/sbin/sysctl.conf` with the value from the output of the previous command.

```
# The min_free_kbytes setting  
  
vm.min_free_kbytes=5572
```

4. Run `sysctl -p` to apply the changes in `sysctl.conf` immediately.

Note: These steps will need to be replicated for each node in the cluster.

User Max Open Files Limit

This topic details how to change the user max open-files limit setting to meet Vertica requirements. The installer reports this issue with the identifier: **S0060**.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

Vertica requires that the `dbadmin` user not be limited when opening files. The open file limit should be at least 1 file open per MB of RAM, 65536, or the amount of RAM in MB; whichever is greater. Vertica sets this to the minimum recommended value of 65536 or the amount of RAM in MB.

All Systems

The open file limit can be determined by running `ulimit -n` as the `dbadmin` user. For example:

```
dbadmin@localhost:$ ulimit -n
65536
```

To manually set the limit, edit `/etc/security/limits.conf` and edit/add the line for the `nofile` setting for the user you configured as the database admin (default `dbadmin`). The setting must be at least 65536.

```
dbadmin -          nofile 65536
```

Note: There is also an open file limit on the system. See [System Max Open Files Limit](#).

System Max Open Files Limit

This topic details how to modify the limit for the number of open files on your system so that it meets Vertica requirements. The installer reports this issue with the identifier: **S0120**.

Vertica opens many files. Some platforms have global limits on the number of open files. The open file limit must be set sufficiently high so as not to interfere with database operations.

The recommended value is at least the amount of memory in MB, but not less than 65536.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To manually set the open file limit:

1. Run `/sbin/sysctl fs.file-max` to determine the current limit.
2. If the limit is not **65536** or the amount of system memory in MB (whichever is higher), then edit or add `fs.file-max=max number of files` to `/etc/sysctl.conf`.

```
# Controls the maximum number of open files
fs.file-max=65536
```

3. Run `sysctl -p` to apply the changes in `sysctl.conf` immediately.

Note: These steps will need to be replicated for each node in the cluster.

Pam Limits

This topic details how to enable the "su" `pam_limits.so` module required by Vertica. The installer reports issues with the setting with the identifier: **S0070**.

On some systems the pam module called `pam_limits.so` is not set in the file `/etc/pam.d/su`. When it is not set, it prevents the conveying of limits (such as open file descriptors) to any command started with `su -`.

In particular, the Vertica init script would fail to start Vertica because it calls the Administration Tools to start a database with the `su -` command. This problem was first noticed on Debian systems, but the configuration could be missing on other Linux distributions. See the [pam_limits](#) man page for more details.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To manually configure this setting, append the following line to the `/etc/pam.d/su` file:

```
session required pam_limits.so
```

See the `pam_limits` man page for more details: `man pam_limits`.

pid_max Setting

This topic explains how to change `pid_max` to a supported value. The value of `pid_max` should be

```
pid_max = num_user_proc + 2**15 = num_user_proc + 32768
```

where `num_user_proc` is the size of memory in megabytes.

The minimum value for `pid_max` is 524288.

If your `pid_max` value is too low, the installer reports this problem and indicates the minimum value.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To change the `pid_max` value:

```
# sysctl -w kernel.pid_max=524288
```

User Address Space Limits

This topic details how to modify the Linux address space limit for the `dbadmin` user so that it meets Vertica requirements. The address space setting controls the maximum number of threads and processes for each user. If this setting does not meet the requirements then the installer reports this issue with the identifier: **S0090**.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

The address space available to the `dbadmin` user must not be reduced via user limits and must be set to **unlimited**.

All Systems

To manually set the address space limit:

1. Run `ulimit -v` as the `dbadmin` user to determine the current limit.
2. If the limit is not **unlimited**, then add the following line to `/etc/security/limits.conf`. Replace `dbadmin` with your database admin user

```
dbadmin - as unlimited
```

User File Size Limit

This topic details how to modify the file size limit for files on your system so that it meets Vertica requirements. The installer reports this issue with the identifier: **S0100**.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

The file size limit for the `dbadmin` user must not be reduced via user limits and must be set to **unlimited**.

All Systems

To manually set the file size limit:

1. Run `ulimit -f` as the `dbadmin` user to determine the current limit.
2. If the limit is not **unlimited**, then edit/add the following line to `/etc/security/limits.conf`. Replace `dbadmin` with your database admin user.

```
dbadmin -      fsize      unlimited
```

User Process Limit

This topic details how to change the user process limit so that it meets Vertica requirements. The installer reports this issue with the identifier: **S0110**.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

The user process limit must be high enough to allow for the many threads opened by Vertica. The recommended limit is the amount of RAM in MB and must be at least 1024.

All Systems

To manually set the user process limit:

1. Run `ulimit -u` as the `dbadmin` user to determine the current limit.
2. If the limit is not the amount of memory in MB on the server, then edit/add the following line to `/etc/security/limits.conf`. Replace `4096` with the amount of system memory, in MB, on the server.

```
dbadmin - nproc 4096
```

Maximum Memory Maps Configuration

This topic details how to modify the limit for the number memory maps a process can have on your system so that it meets Vertica requirements. The installer reports this issue with the identifier: **S0130**.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

Vertica uses a lot of memory while processing and can approach the default limit for memory maps per process.

The recommended value is at least the amount of memory on the system in KB / 16, but not less than 65536.

All Systems

To manually set the memory map limit:

1. Run `/sbin/sysctl vm.max_map_count` to determine the current limit.
2. If the limit is not **65536** or the amount of system memory in KB / 16 (whichever is higher), then edit/add the following line to `/etc/sysctl.conf`. Replace `65536` with the value for your system.

```
# The following 1 line added by Vertica tools. 2014-03-07 13:20:31  
vm.max_map_count=65536
```

3. Run `sysctl -p` to apply the changes in `sysctl.conf` immediately.

Note: These steps will need to be replicated for each node in the cluster.

General Operating System Configuration - Manual Configuration

The following general Operating System settings must be done manually.

Manually Configuring Operating System Settings

Vertica requires that you manually configure some general operating system settings. Vertica recommends that you configure these settings in the `/etc/rc.local` script to prevent them from reverting on reboot. The `/etc/rc.local` startup script contains scripts and commands that run each time the system is booted.

If you are using Red Hat 7.0 or CentOS 7.0 or higher, you must make sure the tuning system service will not start upon reboot. Run the following command as `sudo` or `root`:

```
$ chkconfig tuned off
```

Turning off tuning prevents monitoring of your OS and any tuning of your OS based on this monitoring. Tuning also enables THP silently which may cause issues in other areas, for example read ahead.

Note: SUSE systems use the `/etc/init.d/after.local` file rather than the `etc/rc.local` file. For purposes of using Vertica, the functionality of both files is the same.

Settings to Configure Manually

The `/etc/rc.local` settings relevant to the installation of Vertica include:

- [Disk Readahead](#)
- [I/O Scheduling](#)
- [Enabling or Disabling Transparent Hugepages](#)

Permanently Changing Settings with /etc/rc.local

1. As the root user, open the /etc/rc.local file:

```
# vi /etc/rc.local
```

2. Enter a script or command. For example, to set the transparent hugepages setting to meet Vertica requirements, enter:

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

Important: On some Ubuntu/Debian systems, the last line in /etc/rc.local must be "exit 0". Any additions to /etc/rc.local must come before "exit 0".

3. Save your changes, and close the /etc/rc.local file.
4. If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

On the next reboot, the command runs during startup. You can also run the command manually, as the root user, if you want it to take effect immediately.

Check for Swappiness

The swappiness kernel parameter defines the amount, and how often, the kernel copies RAM contents to a swap space. Vertica recommends a value of 1. The installer reports any swappiness issues with identifier **S0112**.

To set the swappiness value add or update the following in /etc/sysctl.conf:

```
vm.swappiness = 1
```

This also ensures that the value persists after a reboot.

You can check the swappiness value as follows:

```
cat /proc/sys/vm/swappiness
```

If necessary, you change the swappiness value at runtime by logging in as root and running the following:

```
echo 1 > /proc/sys/vm/swappiness
```

Disk Readahead

This topic details how to change [Disk Readahead](#) to a supported value. Vertica requires that Disk Readahead be set to at least 2048. The installer reports this issue with the identifier: **S0020**.

Note:

- These commands must be executed with root privileges and assumes the blockdev program is in `/sbin`.
- The blockdev program operates on whole devices, and not individual partitions. You cannot set the readahead value to different settings on the same device. If you run blockdev against a partition, for example: `/dev/sda1`, then the setting is still applied to the entire `/dev/sda` device. For instance, running `/sbin/blockdev --setra 2048 /dev/sda1` also causes `/dev/sda2 through /dev/sdaN` to use a readahead value of 2048.

RedHat/CentOS and SuSE Based Systems

For each drive in the Vertica system, Vertica recommends that you set the readahead value to at least 2048 for most deployments. The command immediately changes the readahead value for the specified disk. The second line adds the command to `/etc/rc.local` so that the setting is applied each time the system is booted. Note that some deployments may require a higher value and the setting can be set as high as 8192, under guidance of support.

Note: For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

```
/sbin/blockdev --setra 2048 /dev/sda  
echo '/sbin/blockdev --setra 2048 /dev/sda' >> /etc/rc.local
```

If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

Ubuntu and Debian Systems

For each drive in the Vertica system, set the readahead value to 2048. Run the command once in your shell, then add the command to `/etc/rc.local` so that the setting is applied each time the system is booted. Note that on Ubuntu systems, the last line in `rc.local` must be `exit 0`. So you must manually add the following line to `etc/rc.local` before the last line with `exit 0`.

Note: For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

```
/sbin/blockdev --setra 2048 /dev/sda
```

Enabling Network Time Protocol (NTP)

The network time protocol (NTP) daemon must be running on all of the hosts in the cluster so that their clocks are synchronized. The spread daemon relies on all of the nodes to have their clocks synchronized for timing purposes. If your nodes do not have NTP running, the installation can fail with a spread configuration error or other errors.

Note: Different Linux distributions refer to the NTP daemon in different ways. For example, SUSE and Debian/Ubuntu refer to it as `ntp`, while CentOS and Red Hat refer to it as `ntpd`. If the following commands produce errors, try using `ntp` in place of `ntpd`.

Verify That NTP Is Running

To verify that your hosts are configured to run the NTP daemon on startup, enter the following command:

```
$ chkconfig --list ntpd
```

Debian and Ubuntu do not support `chkconfig`, but they do offer an optional package. You can install this package with the command `sudo apt-get install sysv-rc-conf`. To verify that your hosts are configured to run the NTP daemon on startup with the `sysv-rc-conf` utility, enter the following command:

```
$ sysv-rc-conf --list ntpd
```

The `chkconfig` command can produce an error similar to `ntpd: unknown service`. If you get this error, verify that your Linux distribution refers to the NTP daemon as `ntpd` rather than `ntp`. If it does not, you need to install the NTP daemon package before you can configure it. Consult your Linux documentation for instructions on how to locate and install packages.

If the NTP daemon is installed, your output should resemble the following:

```
ntp 0:off 1:off 2:on 3:on 4:off 5:on 6:off
```

The output indicates the runlevels where the daemon runs. Verify that the current runlevel of the system (usually 3 or 5) has the NTP daemon set to `on`. If you do not know the current runlevel, you can find it using the `runlevel` command:

```
$ runlevel
N 3
```

Configure NTP for Red Hat 6/CentOS 6 and SLES

If your system is based on Red Hat 6/CentOS 6 or SUSE Linux Enterprise Server, use the `service` and `chkconfig` utilities to start NTP and have it start at startup.

```
/sbin/service ntpd restart
/sbin/chkconfig ntpd on
```

- **Red Hat 6/CentOS 6**—NTP uses the default time servers at `ntp.org`. You can change the default NTP servers by editing `/etc/ntp.conf`.
- **SLES**—By default, no time servers are configured. You must edit `/etc/ntp.conf` after the install completes and add time servers.

Configure NTP for Ubuntu and Debian

By default, the [NTP daemon](#) is not installed on some Ubuntu and Debian systems. First, install NTP, and then start the NTP process. You can change the default NTP servers by editing `/etc/ntp.conf` as shown:

```
sudo apt-get install ntp
sudo /etc/init.d/ntp reload
```

Verify That NTP Is Operating Correctly

To verify that the Network Time Protocol Daemon (NTPD) is operating correctly, issue the following command on all nodes in the cluster.

For Red Hat 6/CentOS 6 and SLES:

```
/usr/sbin/ntpq -c rv | grep stratum
```

For Ubuntu and Debian:

```
ntpq -c rv | grep stratum
```

A stratum level of 16 indicates that NTP is not synchronizing correctly.

If a stratum level of 16 is detected, wait 15 minutes and issue the command again. It may take this long for the NTP server to stabilize.

If NTP continues to detect a stratum level of 16, verify that the NTP port (UDP Port 123) is open on all firewalls between the cluster and the remote machine to which you are attempting to synchronize.

Red Hat Documentation Related to NTP

The preceding links were current as of the last publication of the Vertica documentation and could change between releases.

- <http://kbase.redhat.com/faq/docs/DOC-6731>
- <http://kbase.redhat.com/faq/docs/DOC-6902>
- <http://kbase.redhat.com/faq/docs/DOC-6991>

Enabling chrony or ntpd for Red Hat 7/CentOS 7 Systems

Before you can install Vertica, you must enable one of the following on your system for clock synchronization:

- chrony
- NTPD

You must enable and activate the Network Time Protocol (NTP) before installation. Otherwise, the installer reports this issue with the identifier **S0030**.

For information on installing and using chrony, see the information below. For information on NTPD see [Enabling Network Time Protocol \(NTP\)](#).

Install chrony

The chrony suite consists of:

- `chronyd` - the daemon for clock synchronization.
- `chronyc` - the command-line utility for configuring `chronyd` .

`chrony` is installed by default on some versions of Red Hat/CentOS 7. However, if `chrony` is not installed on your system, you must download it. To download `chrony`, run the following command as `sudo` or `root`:

```
# yum install chrony
```

Verify That `chrony` Is Running

To view the status of the `chronyd` daemon, run the following command:

```
$ systemctl status chronyd
```

If `chrony` is running, an output similar to the following appears:

```
chronyd.service - NTP client/server
  Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled)
  Active: active (running) since Mon 2015-07-06 16:29:54 EDT; 15s ago
  Main PID: 2530 (chronyd)
  CGroup: /system.slice/chronyd.service
          ┆â2530 /usr/sbin/chronyd -u chrony
```

If `chrony` is not running, execute the following command as `sudo` or `root`. This command also causes `chrony` to run at boot time:

```
# systemctl enable chronyd
```

Verify That `chrony` Is Operating Correctly

To verify that the `chrony` daemon is operating correctly, issue the following command on all nodes in the cluster:

```
$ chronyc tracking
```

An output similar to the following appears:

```
Reference ID      : 198.247.63.98 (time01.website.org)
Stratum          : 3
Ref time (UTC)   : Thu Jul  9 14:58:01 2015
System time      : 0.000035685 seconds slow of NTP time
Last offset      : -0.000151098 seconds
RMS offset       : 0.000279871 seconds
Frequency        : 2.085 ppm slow
Residual freq    : -0.013 ppm
```

```
Skew          : 0.185 ppm
Root delay    : 0.042370 seconds
Root dispersion : 0.022658 seconds
Update interval : 1031.0 seconds
Leap status   : Normal
```

A stratum level of 16 indicates that chrony is not synchronizing correctly. If chrony continues to detect a stratum level of 16, verify that the UDP port 323 is open. This port must be open on all firewalls between the cluster and the remote machine to which you are attempting to synchronize.

Red Hat Documentation Related to chrony

These links to Red Hat documentation were current as of the last publication of the Vertica documentation. Be aware that they could change between releases:

- [Configuring NTP Using the chrony Suite](#)
- [Using chrony](#)

SELinux Configuration

Vertica does not support SELinux except when SELinux is running in permissive mode. If it detects that SELinux is installed and the mode cannot be determined the installer reports this issue with the identifier: **S0080**. If the mode can be determined, and the mode is not permissive, then the issue is reported with the identifier: **S0081**.

Red Hat and SUSE Systems

You can either disable SELinux or change it to use permissive mode.

To disable SELinux:

1. Edit `/etc/selinux/config` and change setting for SELinux to disabled (`SELINUX=disabled`). This disables SELinux at boot time.
2. As root/sudo, type `setenforce 0` to disable SELinux immediately.

To change SELinux to use permissive mode:

1. Edit `/etc/selinux/config` and change setting for SELINUX to permissive (`SELINUX=Permissive`).

2. As root/sudo, type `setenforce Permissive` to switch to permissive mode immediately.

Ubuntu and Debian Systems

You can either disable SELinux or change it to use permissive mode.

To disable SELinux:

1. Edit `/selinux/config` and change setting for SELinux to disabled (`SELINUX=disabled`). This disables SELinux at boot time.
2. As root/sudo, type `setenforce 0` to disable SELinux immediately.

To change SELinux to use permissive mode:

1. Edit `/selinux/config` and change setting for SELinux to permissive (`SELINUX=Permissive`).
2. As root/sudo, type `setenforce Permissive` to switch to permissive mode immediately.

CPU Frequency Scaling

This topic details the various CPU frequency scaling methods supported by Vertica. In general, if you do not require CPU frequency scaling, then disable it so as not to impact system performance.

Important: Your systems may use significantly more energy when frequency scaling is disabled.

The installer allows CPU frequency scaling to be enabled when the `cpufreq` scaling governor is set to `performance`. If the `cpu` scaling governor is set to `ondemand`, and `ignore_nice_load` is 1 (true), then the installer **fails** with the error **S0140**. If the `cpu` scaling governor is set to `ondemand` and `ignore_nice_load` is 0 (false), then the installer **warns** with the identifier **S0141**.

CPU frequency scaling is a hardware and software feature that helps computers conserve energy by slowing the processor when the system load is low, and speeding it up again when the system load increases. This feature can impact system performance, since raising the CPU frequency in response to higher system load does not occur instantly. Always disable this feature on the Vertica database hosts to prevent it from interfering with performance.

You disable CPU scaling in your host's system BIOS. There may be multiple settings in your host's BIOS that you need to adjust in order to completely disable CPU frequency scaling. Consult your host hardware's documentation for details on entering the system BIOS and disabling CPU frequency scaling.

If you cannot disable CPU scaling through the system BIOS, you can limit the impact of CPU scaling by disabling the scaling through the Linux kernel or setting the CPU frequency governor to always run the CPU at full speed.

Caution: This method is not reliable, as some hardware platforms may ignore the kernel settings. **The only reliable method is to disable CPU scaling in BIOS.**

The method you use to disable frequency depends on the CPU scaling method being used in the Linux kernel. See your Linux distribution's documentation for instructions on disabling scaling in the kernel or changing the CPU governor.

Enabling or Disabling Transparent Hugepages

You can modify transparent hugepages so that the configuration meets Vertica requirements.

- For Red Hat 7/CentOS 7 systems, you must enable transparent hugepages. The installer reports this issue with the identifier: **S0312**.
- For all other systems, you must disable transparent hugepages or set them to `madvise`. The installer reports this issue with the identifier: **S0310**.

Disable Transparent Hugepages on Red Hat 6/CentOS 6 Systems

Important: If you are using Red Hat 7/CentOS 7, you must enable, rather than disable transparent hugepages. See: [Enable Transparent Hugepages on Red Hat 7/CentOS 7 Systems](#).

Determine if transparent hugepages is enabled. To do so, run the following command.

```
cat /sys/kernel/mm/redhat_transparent_hugepage/enabled  
[always] madvise never
```

The setting returned in brackets is your current setting.

If you are not using `madvise` or `never` as your transparent hugepage setting, then you can disable transparent hugepages in one of two ways:

- Edit your boot loader (for example `/etc/grub.conf`). Typically, you add the following to the end of the kernel line. However, consult the documentation for your system before editing your boot loader configuration.

```
transparent_hugepage=never
```

- Edit `/etc/rc.local` and add the following script.

```
if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then
  echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
fi
```

For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

Regardless of which approach you choose, you must reboot your system for the setting to take effect, or run the following echo line to proceed with the install without rebooting:

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

Enable Transparent Hugepages on Red Hat 7/CentOS 7 Systems

Determine if transparent hugepages is enabled. To do so, run the following command.

```
cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

The setting returned in brackets is your current setting.

For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

You can enable transparent hugepages by editing `/etc/rc.local` and adding the following script:

```
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
  echo always > /sys/kernel/mm/transparent_hugepage/enabled
fi
```

You must reboot your system for the setting to take effect, or, as root, run the following echo line to proceed with the install without rebooting:

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

Disable Transparent Hugepages on Other Systems

Note: SuSE did not offer transparent hugepage support in its initial 11.0 release. However, subsequent SuSE service packs do include support for transparent hugepages.

To determine if transparent hugepages is enabled, run the following command.

```
cat /sys/kernel/mm/transparent_hugepage/enabled  
[always] madvise never
```

The setting returned in brackets is your current setting. Depending on your platform OS, the `madvise` setting may not be displayed.

You can disable transparent hugepages one of two ways:

- Edit your boot loader (for example `/etc/grub.conf`). Typically, you add the following to the end of the kernel line. However, consult the documentation for your system before editing your bootloader configuration.

```
transparent_hugepage=never
```

- Edit `/etc/rc.local` (on systems that support `rc.local`) and add the following script.

```
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then  
    echo never > /sys/kernel/mm/transparent_hugepage/enabled  
fi
```

For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

Regardless of which approach you choose, you must reboot your system for the setting to take effect, or run the following two echo lines to proceed with the install without rebooting:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Disabling Defrag for Red Hat and CentOS Systems

On all Red Hat and CentOS systems, you must disable the defrag utility to meet Vertica configuration requirements. The steps necessary to disable defrag on Red Hat 6/CentOS 6 systems differ from those used to disable defrag on Red Hat 7/CentOS 7 systems.

Disable Defrag on Red Hat 6/CentOS 6 Systems

1. Determine if defrag is enabled by running the following command:

```
cat /sys/kernel/mm/redhat_transparent_hugepage/defrag  
[always] madvise never
```

The setting returned in brackets is your current setting. If you are not using `madvise` or `never` as your defrag setting, then you must disable defrag.

2. Edit `/etc/rc.local`, and add the following script:

```
if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then  
    echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag  
fi
```

You must reboot your system for the setting to take effect, or run the following echo line to proceed with the install without rebooting:

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

Disable Defrag on Red Hat 7/CentOS 7 Systems

1. Determine if defrag is enabled by running the following command:

```
cat /sys/kernel/mm/transparent_hugepage/defrag  
[always] madvise never
```

The setting returned in brackets is your current setting. If you are not using `madvise` or `never` as your defrag setting, then you must disable defrag.

2. Edit `/etc/rc.local`, and add the following script:

```
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then  
    echo never > /sys/kernel/mm/transparent_hugepage/defrag  
fi
```

You must reboot your system for the setting to take effect, or run the following echo line to proceed with the install without rebooting:

```
# echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

3. If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

I/O Scheduling

This topic details how to change [I/O Scheduling](#) to a supported scheduler. Vertica requires that I/O Scheduling be set to [deadline](#) or [noop](#). The installer checks what scheduler the system is using, reporting an unsupported scheduler issue with identifier: **S0150**. If the installer cannot detect the type of scheduler in use (typically if your system is using a RAID array), it reports that issue with identifier: **S0151**.

If your system is not using a RAID array, then complete the following steps to change your system to a supported I/O Scheduler. If you are using a RAID array, then consult your RAID vendor documentation for the best performing scheduler for your hardware.

Configure the I/O Scheduler

The Linux kernel can use several different I/O schedulers to prioritize disk input and output. Most Linux distributions use the Completely Fair Queuing (CFQ) scheme by default, which gives input and output requests equal priority. This scheduler is efficient on systems running multiple tasks that need equal access to I/O resources. However, it can create a bottleneck when used on Vertica drives containing the catalog and data directories, because it gives write requests equal priority to read requests, and its per-process I/O queues can penalize processes making more requests than other processes.

Instead of the CFQ scheduler, configure your hosts to use either the Deadline or NOOP I/O scheduler for the drives containing the catalog and data directories:

- The Deadline scheduler gives priority to read requests over write requests. It also imposes a deadline on all requests. After reaching the deadline, such requests gain priority over all other requests. This scheduling method helps prevent processes from becoming starved for

I/O access. The Deadline scheduler is best used on physical media drives (disks using spinning platters), since it attempts to group requests for adjacent sectors on a disk, lowering the time the drive spends seeking.

- The NOOP scheduler uses a simple FIFO approach, placing all input and output requests into a single queue. This scheduler is best used on solid state drives (SSDs). Because SSDs do not have a physical read head, no performance penalty exists when accessing non-adjacent sectors.

Failure to use one of these schedulers for the Vertica drives containing the catalog and data directories can result in slower database performance. Other drives on the system (such as the drive containing swap space, log files, or the Linux system files) can still use the default CFQ scheduler (although you should always use the NOOP scheduler for SSDs).

There are two ways for you to set the scheduler used by your disk devices:

1. Write the name of the scheduler to a file in the `/sys` directory.

--or--

2. Use a kernel boot parameter.

Configure the I/O Scheduler - Changing the Scheduler Through the `/sys` Directory

You can view and change the scheduler Linux uses for I/O requests to a single drive using a virtual file under the `/sys` directory. The name of the file that controls the scheduler a block device uses is:

```
/sys/block/deviceName/queue/scheduler
```

Where *deviceName* is the name of the disk device, such as `sda` or `cciss\!c0d1` (the first disk on an Micro Focus RAID array). Viewing the contents of this file shows you all of the possible settings for the scheduler. The currently-selected scheduler is surrounded by square brackets:

```
# cat /sys/block/sda/queue/scheduler  
noop deadline [cfq]
```

To change the scheduler, write the name of the scheduler you want the device to use to its scheduler file. You must have root privileges to write to this file. For example, to set the `sda` drive to use the deadline scheduler, run the following command as root:

```
# echo deadline > /sys/block/sda/queue/scheduler  
# cat /sys/block/sda/queue/scheduler
```

```
noop [deadline] cfq
```

Changing the scheduler immediately affects the I/O requests for the device. The Linux kernel starts using the new scheduler for all of the drive's input and output requests.

Note: While tests show that changing the scheduler settings while Vertica is running does not cause problems, Vertica recommends shutting down. Before changing the I/O schedule, or making any other changes to the system configuration, consider shutting down any running database.

Changes to the I/O scheduler made through the `/sys` directory only last until the system is rebooted, so you need to add the commands that change the I/O scheduler to a startup script (such as those stored in `/etc/init.d`, or through a command in `/etc/rc.local`). You also need to use a separate command for each drive on the system whose scheduler you want to change.

For example, to make the configuration take effect immediately and add it to `rc.local` so it is used on subsequent reboots.

Note: For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

```
echo deadline > /sys/block/sda/queue/scheduler  
echo 'echo deadline > /sys/block/sda/queue/scheduler' >> /etc/rc.local
```

Note: On some Ubuntu/Debian systems, the last line in `rc.local` must be `exit 0`. So you must manually add the following line to `etc/rc.local` before the last line with `exit 0`.

You may prefer to use this method of setting the I/O scheduler over using a boot parameter if your system has a mix of solid-state and physical media drives, or has many drives that do not store Vertica catalog and data directories.

If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

Configure the I/O Scheduler - Changing the Scheduler with a Boot Parameter

Use the `elevator` kernel boot parameter to change the default scheduler used by all disks on your system. This is the best method to use if most or all of the drives on your hosts are of the same type (physical media or SSD) and will contain catalog or data files. You can also use the

boot parameter to change the default to the scheduler the majority of the drives on the system need, then use the `/sys` files to change individual drives to another I/O scheduler. The format of the elevator boot parameter is:

```
elevator=schedulerName
```

Where *schedulerName* is `deadline`, `noop`, or `cfq`. You set the boot parameter using your bootloader (`grub` or `grub2` on most recent Linux distributions). See your distribution's documentation for details on how to add a kernel boot parameter.

Support Tools

Vertica suggests that the following tools are installed so support can assist in troubleshooting your system if any issues arise:

- `pstack` (or `gstack`) package. Identified by issue **S0040** when not installed.
 - On Red Hat 7 and CentOS 7 systems, the `pstack` package is installed as part of the `gdb` package.
- `mcelog` package. Identified by issue **S0041** when not installed.
- `sysstat` package. Identified by issue **S0045** when not installed.

Red Hat 6 and CentOS 6 Systems

To install the required tools on Red Hat 6 and CentOS 6 systems, run the following commands as `sudo` or `root`:

```
yum install pstack  
yum install mcelog  
yum install sysstat
```

Red Hat 7 and CentOS 7 Systems

To install the required tools on Red Hat 7/CentOS 7 systems, run the following commands as `sudo` or `root`:

```
yum install gdb  
yum install mcelog  
yum install sysstat
```

Ubuntu and Debian Systems

To install the required tools on Ubuntu and Debian systems, run the following commands as `sudo` or `root`:

```
apt-get install pstack  
apt-get install mcelog  
apt-get install sysstat
```

SuSE Systems

To install the required tools on SuSE systems, run the following commands as `sudo` or `root`.

```
zypper install sysstat  
zypper install mcelog
```

There is no individual SuSE package for `pstack/gstack`. However, the `gdb` package contains `gstack`, so you could optionally install `gdb` instead, or build `pstack/gstack` from source. To install the `gdb` package:

```
zypper install gdb
```

System User Configuration

The following tasks pertain to the configuration of the system user required by Vertica.

System User Requirements

Vertica has specific requirements for the system user that runs and manages Vertica. If you specify a user during install, but the user does not exist, then the installer reports this issue with the identifier: **S0200**.

System User Requirement Details

Vertica requires a system user to own database files and run database processes and administration scripts. By default, the install script automatically configures and creates this user for you with the username `dbadmin`. See [About Linux Users Created by Vertica and Their Privileges](#) for details on the default user created by the install script. If you decide to manually

create your own system user, then you must create the user **before** you run the install script. If you manually create the user:

Note: Instances of `dbadmin` and `verticadba` are placeholders for the names you choose if you do not use the default values.

- the user must have the same username and password on all nodes
- the user must use the BASH shell as the user's default shell. If not, then the installer reports this issue with identifier **[S0240]**.
- the user must be in the `verticadba` group (for example: `usermod -a -G verticadba userNameHere`). If not, the installer reports this issue with identifier **[S0220]**.

Note: You must create a `verticadba` group on all nodes. If you do not, then the installer reports the issue with identifier **[S0210]**.

- the user's login group must be either `verticadba` or a group with the same name as the user (for example, the home group for `dbadmin` is `dbadmin`). You can check the groups for a user with the `id` command. For example: `id dbadmin`. The "gid" group is the user's primary group. If this is not configured correctly then the installer reports this issue with the identifier **[S0230]**. Vertica recommends that you use `verticadba` as the user's primary login group. For example: `usermod -g verticadba userNameHere`. If the user's primary group is not `verticadba` as suggested, then the installer reports this with HINT **[S0231]**.
- the user must have a home directory. If not, then the installer reports this issue with identifier **[S0260]**.
- the user's home directory must be owned by the user. If not, then the installer reports the issue with identifier **[S0270]**.
- the system must be aware of the user's home directory (you can set it with the `usermod` command: `usermod -m -d /path/to/new/home/dir userNameHere`). If this is not configured correctly then the installer reports the issue with **[S0250]**.
- the user's home directory must be owned by the `dbadmin`'s primary group (use the `chown` and `chgrp` commands if necessary). If this is not configured correctly, then the installer reports the issue with identifier **[S0280]**.
- the user's home directory *should* have secure permissions. Specifically, it should not be writable by anyone or by the group. Ideally the permissions should be, when viewing with `ls`, `"---"` (nothing), or `"r-x"` (read and execute). If this is not configured as suggested then the installer reports this with HINT **[S0290]**.

TZ Environment Variable

This topic details how to set or change the TZ environment variable and update your tzdata package. If this variable is not set, then the installer reports this issue with the identifier: **S0305**.

Before installing Vertica, update the tzdata package for your system and set the default time zone for your database administrator account by specifying the TZ environmental variable. If your database administrator is being created by the `install_vertica` script, then set the TZ variable after you have installed Vertica.

Update tzdata Package

The tzdata package is a public-domain time zone database that is pre-installed on most Linux systems. The tzdata package is updated periodically for time-zone changes across the world. Micro Focus recommends that you update to the latest tzdata package before installing or updating Vertica.

Update your tzdata package with the following command:

- RedHat based systems: `yum update tzdata`
- Debian and Ubuntu systems: `apt-get install tzdata`

Setting the Default Time Zone

When a client receives the result set of a SQL query, all rows contain data adjusted, if necessary, to the same time zone. That time zone is the default time zone of the initiator node unless the client explicitly overrides it using the SQL [SET TIME ZONE](#) command described in the SQL Reference Manual. The default time zone of any node is controlled by the TZ environment variable. If TZ is undefined, the operating system time zone.

Important: The TZ variable must be set to the same value on all nodes in the cluster.

If your operating system timezone is not set to the desired timezone of the database then make sure that the Linux environment variable TZ is set to the desired value on all cluster hosts.

The installer returns a warning if the TZ variable is not set. If your operating system timezone is appropriate for your database, then the operating system timezone is used and the warning can be safely ignored.

Setting the Time Zone on a Host

Important: If you explicitly set the TZ environment variable at a command line before you start the Administration Tools, the current setting will not take effect. The Administration Tools uses SSH to start copies on the other nodes, so each time SSH is used, the TZ variable for the startup command is reset. TZ must be set in the `.profile` or `.bashrc` files on all nodes in the cluster to take affect properly.

You can set the time zone several different ways, depending on the Linux distribution or the system administrator's preferences.

- To set the system time zone on Red Hat and SUSE Linux systems, edit:

```
/etc/sysconfig/clock
```

- To set the TZ variable, edit, `/etc/profile`, or `/dbadmin/.bashrc` or `/home/dbadmin/.bash_profile` and add the following line (for example, for the US Eastern Time Zone):

```
export TZ="America/New_York"
```

For details on which timezone names are recognized by Vertica, see the appendix: [Using Time Zones With Vertica](#).

LANG Environment Variable Settings

This topic details how to set or change the LANG environment variable. The LANG environment variable controls the locale of the host. If this variable is not set, then the installer reports this issue with the identifier: **S0300**. If this variable is not set to a valid value, then the installer reports this issue with the identifier: **S0301**.

Set the Host Locale

Each host has a system setting for the Linux environment variable LANG. LANG determines the locale category for native language, local customs, and coded character set in the absence of the LC_ALL and other LC_ environment variables. LANG can be used by applications to determine which language to use for error messages and instructions, collating sequences, date formats, and so forth.

To change the `LANG` setting for the database administrator, edit `/etc/profile`, or `/dbadmin/.bashrc` or `/home/dbadmin/.bash_profile` on all cluster hosts and set the environment variable; for example:

```
export LANG=en_US.UTF8
```

The `LANG` setting controls the following in Vertica:

- OS-level errors and warnings, for example, "file not found" during [COPY](#) operations.
- Some formatting functions, such as [TO_CHAR](#) and [TO_NUMBER](#). See also [Template Patterns for Numeric Formatting](#).

The `LANG` setting does not control the following:

- Vertica specific error and warning messages. These are always in English at this time.
- Collation of results returned by SQL issued to Vertica. This must be done using a database parameter instead. See [Implement Locales for International Data Sets](#) section in the Administrator's Guide for details.

Note: If the `LC_ALL` environment variable is set, it supersedes the setting of `LANG`.

Package Dependencies

For successful Vertica installation, you must first install three packages on all nodes in your cluster before installing the database platform.

The required packages are:

- `openssh`—Required for Administration Tools connectivity between nodes.
- `which`—Required for Vertica operating system integration and for validating installations.
- `dialog`—Required for interactivity with Administration Tools.

Installing the Required Packages

The procedure you follow to install the required packages depends on the operating system on which your node or cluster is running. See your operating system's documentation for detailed information on installing packages.

- **For CentOS/Red Hat Systems**—Typically, you manage packages on Red Hat and CentOS systems using the yum utility.

Run the following yum commands to install each of the package dependencies. The yum utility guides you through the installation:

```
# yum install openssh  
# yum install which  
# yum install dialog
```

- **For Debian/Ubuntu Systems**—Typically, you use the apt-get utility to manage packages on Debian and Ubuntu systems.

Run the following apt-get commands to install each of the package dependencies. The apt-get utility guides you through the installation:

```
# apt-get install openssh  
# apt-get install which  
# apt-get install dialog
```

Installing Vertica

There are different paths you can take when installing Vertica. You can:

- Install Vertica on one or more hosts using the command line, and not use the Management Console.
- Install the Management Console, and from the Management Console install Vertica on one or more hosts by using the Management Console cluster creation wizard.
- Install Vertica on one or more hosts using the command line, then install the Management Console and import the cluster to be managed.

Installing Using the Command Line

Although Micro Focus supports installation on one node, two nodes, and multiple nodes, this section describes how to install the Vertica software on a cluster of nodes. It assumes that you have already performed the tasks in [Before You Install Vertica](#), and that you have a Vertica license key.

To install Vertica, complete the following tasks:

1. [Download and install the Vertica server package](#)
2. [Installing Vertica with the Installation Script](#)

Special notes

- Downgrade installations are not supported.
- Be sure that you download the RPM for the correct operating system and architecture.
- Vertica supports two-node clusters with zero fault tolerance (K=0 safety). This means that you can [add a node](#) to a single-node cluster, as long as the installation node (the node upon which you build) is not the loopback node (`localhost/127.0.0.1`).
- The Version 7.0 installer introduces new platform verification tests that prevent the install from continuing if the platform requirements are not met by your system. Manually verify that your system meets the requirements in [Before You Install Vertica](#) on your systems. These tests ensure that your platform meets the hardware and software

requirements for Vertica. Previous versions documented these requirements, but the installer did not verify all of the settings. If this is a fresh install, then you can simply run the installer and view a list of the failures and warnings to determine which configuration changes you must make.

Download and Install the Vertica Server Package

To download and install the Vertica server package:

1. Use a Web browser to log in to [myVertica portal](#).
2. Click the Download tab and download the Vertica server package to the Administration Host.

Be sure the package you download matches the operating system and the machine architecture on which you intend to install it. In the event of a node failure, you can use any other node to run the Administration Tools later.

3. If you installed a previous version of Vertica on any of the hosts in the cluster, use the Administration Tools to shut down any running database.

The database must stop normally; you cannot upgrade a database that requires recovery.

4. If you are using sudo, skip to the next step. If you are root, log in to the Administration Host as root (or log in as another user and switch to root).

```
$ su - root
password: root-password
#
```

Caution: When installing Vertica using an existing user as the dba, you must exit all UNIX terminal sessions for that user after setup completes and log in again to ensure that group privileges are applied correctly.

After Vertica is installed, you no longer need root privileges. To verify sudo, see [General Hardware and OS Requirements and Recommendations](#).

5. Use one of the following commands to run the RPM package installer:

- If you are root and installing an RPM:

```
# rpm -Uvh pathname
```

- If you are using sudo and installing an RPM:

```
$ sudo rpm -Uvh pathname
```

- If you are using Debian:

```
$ sudo dpkg -i pathname
```

where *pathname* is the Vertica package file you downloaded.

Note: If the package installer reports multiple dependency problems, or you receive the error "*ERROR: You're attempting to install the wrong RPM for this operating system*", then you are trying to install the wrong Vertica server package. Make sure that the machine architecture (32-bit or 64-bit) of the package you downloaded matches the operating system.

Installing Vertica with the Installation Script

You run the install script after you install the Vertica package. The install script is run on a single node, using a Bash shell, and it copies the Vertica package to all other hosts (identified by the `--hosts` argument) in your planned cluster.

The install script runs several tests on each of the target hosts to verify that the hosts meet the system and performance requirements for a Vertica node. The install script modifies some operating system configuration settings to meet these requirements. Other settings cannot be modified by the install script and must be manually reconfigured.

The installation script takes the following basic parameters:

- A list of hosts on which to install.
- Optionally, the Vertica RPM/DEB path and package file name if you have not pre-installed the server package on other potential hosts in the cluster.
- Optionally, a system user name. If you do not provide a user name, then the install script creates a new system user named `dbadmin`. If you do provide a username and the username does not exist on the system, then the install script creates that user.

For example:

```
# /opt/vertica/sbin/install_vertica --hosts node0001,node0002,node0003 \  
--rpm /tmp/vertica_8.1.x.x86_64.RHEL6.rpm \  
--dba-user mydba
```

Note: The install script sets up passwordless ssh for the administrator user across all the hosts. If passwordless ssh is already set up, the install script verifies that it is functioning correctly.

Basic Installation

1. As root (or sudo) run the install script. The script must be run by a BASH shell as root or as a user with sudo privileges. You can configure many options when running the install script. See [install_vertica Options](#) below for the complete list of options.

If the installer fails due to any requirements not being met, you can correct the issue and then run the installer again with the same command line options.

To perform a basic installation:

- As root:

```
# /opt/vertica/sbin/install_vertica --hosts host_list --rpm package_name --dba-user dba_  
username
```

- Using sudo:

```
$ sudo /opt/vertica/sbin/install_vertica --hosts host_list --rpm package_name --dba-user dba_  
username
```

Basic Installation Parameters

Option	Description
<code>--hosts <i>host_list</i></code>	A comma-separated list of IP addresses to include in the cluster; do not include space characters in the list. Examples: <pre>--hosts 127.0.0.1 --hosts 192.168.233.101,192.168.233.102,192.168.233.103</pre>

Option	Description
	<p>Note: Vertica stores only IP addresses in its configuration files. You can provide a hostname to the <code>--hosts</code> parameter, but it is immediately converted to an IP address when the script is run.</p>
<pre>--rpm package_name --deb package_name</pre>	<p>The path and name of the Vertica RPM package. Example:</p> <pre>--rpm /tmp/vertica_8.1.x.x86_64.RHEL6.rpm</pre> <p>For Debian and Ubuntu installs, provide the name of the Debian package, for example:</p> <pre>--deb /tmp/vertica_7.2.x86.deb</pre>
<pre>--dba-user dba_username</pre>	<p>The name of the Database Administrator system account to create. Only this account can run the Administration Tools. If you omit the <code>--dba-user</code> parameter, then the default database administrator account name is <code>dbadmin</code>.</p> <p>This parameter is optional for new installations done as root but must be specified when upgrading or when installing using <code>sudo</code>. If upgrading, use the <code>-u</code> parameter to specify the same DBA account name that you used previously. If installing using <code>sudo</code>, the user must already exist.</p> <p>Note: If you manually create the user, modify the user's <code>.bashrc</code> file to include the line: <code>PATH=/opt/vertica/bin:\$PATH</code> so that the Vertica tools such as <code>vsq</code> and <code>admintools</code> can be easily started by the <code>dbadmin</code> user.</p>

- When prompted for a password to log into the other nodes, provide the requested password. Doing so allows the installation of the package and system configuration on the other cluster nodes.
 - If you are root, this is the root password.
 - If you are using `sudo`, this is the sudo user password.

The password does not echo on the command line. For example:

```
Vertica Database 7.0 Installation Tool  
Please enter password for root@host01:password
```

3. If the dbadmin user, or the user specified in the argument `--dba-user`, does not exist, then the install script prompts for the password for the user. Provide the password. For example:

```
Enter password for new UNIX user dbadmin:password  
Retype new UNIX password for user dbadmin:password
```

4. Carefully examine any warnings or failures returned by `install_vertica` and correct the problems.

For example, insufficient RAM, insufficient network throughput, and too high readahead settings on the filesystem could cause performance problems later on. Additionally, LANG warnings, if not resolved, can cause database startup to fail and issues with VSQL. The system LANG attributes must be UTF-8 compatible. **Once you fix the problems, re-run the install script.**

5. When installation is successful, disconnect from the Administration Host, as instructed by the script. Then, complete the required post-installation steps.

At this point, root privileges are no longer needed and the database administrator can perform any remaining steps.

Install on a FIPS 140-2 Enabled Machine

Vertica supports the implementation of the Federal Information Processing Standard 140-2 (FIPS). FIPS mode can be enabled in the operating system, specifically Red Hat Enterprise Linux 6.6.

Note: FIPS enablement on the operating system occurs outside of Vertica.

During the installation process, the Vertica 8.0 installer detects whether the host is operating in FIPS mode, as follows:

- The installer searches for `/proc/sys/crypto/fips_enabled` and examines its content. If the file exists and contains a '1', the host is operating in FIPS mode and the following message appears:

```
/proc/sys/crypto/fips_enabled exists and contains '1', this is a FIPS system
```

- The installer then installs symbolic links to the system `libcrypto.so.10` and `libssl.so.10` in `/opt/vertica/lib`.
- If the file does not exist or does not contain a '1', the installer compares the version numbers of the host `libcrypto.so` and `libssl.so` to the library versions in the Vertica package. If the host library file names have a higher version number, they are linked to the system copies. Otherwise, the host uses the libraries from the Vertica package.

Important: The following message appears if the OpenSSL version is 1.0.1# (where # indicates versions f to t) or higher like 1.0.2h

```
No version information available
```

This message indicates that you are using a library different from the one that Vertica was built with. As long as you are using a library that is newer than 1.0.1e (but not yet 1.1), Vertica operates correctly.

Create Symbolic Links to OpenSSL

During Vertica installation the installer determines what library Vertica needs to execute and sets up the following symbolic links in `/opt/vertica/lib`:

- `libssl.so.10`
- `libcrypto.so.10`

These Vertica-created symbolic links are linked to the actual filenames located in the following locations:

- `/lib`
- `/lib64`
- `/usr/lib`
- `/usr/lib64`
- `/lib/x86_64-linux-gnu`
- `/lib/i386-linux-gnu`

- /usr/lib/x86_64-linux-gnu
- /usr/lib/i386-linux-gnu

During processing, the installer places the actual files in /opt/vertica/lib:

- libssl.so.1.0.1e
- libcrypto.so.1.0.1e

If you build an updated OpenSSL, the default names resulting from the OpenSSL build process are:

- libcrypto.so.1.0.0
- libssl.so.1.0.0

In this case, the installer looks at the installed files and determines that these are a newer version than 1.0.1.e. When you attempt to compile the libraries, compilation fails.

In order for it to be linked automatically, you can do one of the following:

- Rename the libraries to indicate the version from which it was built. The install script can find it.
- Create new symbolic links in opt/vertica/lib that point to the location of the newer OpenSSL libraries.

Important: Before creating new symbolic links contact your Customer Experience representative.

Create new symbolic links using the instructions provided by your operating system.

Important: If you create new symbolic links, do NOT run install_vertica again. Doing so overwrites the new symbolic links.

For more information see [Implementing FIPS 140-2](#).

Required Post-Installation Steps

1. Log in to the Database Administrator account on the administration host.
2. [Install the License Key](#)

3. Accept the EULA.
4. If you have not already done so, proceed to [Getting Started](#). Otherwise, proceed to [Configuring the Database](#) in the Administrator's Guide.

install_vertica Options

The table below details all options available to the `install_vertica` script. Most options have a long and short form. For example `--hosts` is interchangeable with `-s`. The only required options are `--hosts/-s` and `--rpm/--deb/-r`.

Option	Description
<code>--help</code>	Display help for this script.
<code>--hosts host_list,</code> <code>-s host_list</code>	<p>A comma-separated list of host names or IP addresses to include in the cluster. Do not include spaces in the list. The IP addresses or hostnames must be for unique hosts. You cannot list the same host using multiple IP addresses/hostnames.</p> <p>Examples:</p> <pre>--hosts host01,host02,host03 -s 192.168.233.101,192.168.233.102,192.168.233.103</pre> <p>Note: If you are upgrading an existing installation of Vertica, be sure to use the same host names that you used previously.</p>
<code>--rpm package_name,</code> <code>--deb package_name,</code> <code>-r package_name</code>	<p>The name of the RPM or Debian package. The install package must be provided if you are installing or upgrading multiple nodes and the nodes do not have the latest server package installed, or if you are adding a new node. The <code>install_vertica</code> and <code>update_vertica</code> scripts serially copy the server package to the other nodes and install the package. If you are installing or upgrading a large number of nodes, then consider manually installing the package on all nodes before running the upgrade script, as</p>

Option	Description
	<p>the script runs faster if it does not need to serially upload and install the package on each node.</p> <p>Example:</p> <pre data-bbox="764 436 1401 499">--rpm vertica_8.1.x.x86_64.RHEL6.rpm</pre>
<pre data-bbox="207 531 509 583">--data-dir data_directory, -d data_directory</pre>	<p>The default directory for database data and catalog files. The default is /home/dbadmin.</p> <div data-bbox="764 632 1401 856" style="background-color: #f0f0f0; padding: 5px;"> <p>Note: Do not use a shared directory over more than one host for this setting. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.</p> </div>
<pre data-bbox="207 892 443 919">--temp-dir directory</pre>	<p>The temporary directory used for administrative purposes. If it is a directory within /opt/vertica, then it will be created by the installer. Otherwise, the directory should already exist on all nodes in the cluster. The location should allow dbadmin write privileges.</p> <p>The default is /tmp.</p> <div data-bbox="764 1220 1401 1320" style="background-color: #f0f0f0; padding: 5px;"> <p>Note: This is not a temporary data location for the database.</p> </div>
<pre data-bbox="207 1358 488 1411">--dba-user dba_username, -u dba_username</pre>	<p>The name of the Database Administrator system account to create. Only this account can run the Administration Tools. If you omit the --dba-user parameter, then the default database administrator account name is dbadmin.</p> <p>This parameter is optional for new installations done as root but must be specified when upgrading or when installing using sudo. If upgrading, use the -u parameter to specify the same DBA account name that you used previously. If installing using sudo, the user must already exist.</p>

Option	Description
	<p>Note: If you manually create the user, modify the user's <code>.bashrc</code> file to include the line: <code>PATH=/opt/vertica/bin:\$PATH</code> so that the Vertica tools such as <code>vsq</code> and <code>admintools</code> can be easily started by the <code>dbadmin</code> user.</p>
<code>--dba-group GROUP,</code> <code>-g GROUP</code>	<p>The UNIX group for DBA users. The default is <code>verticadba</code>.</p>
<code>--dba-user-home dba_home_directory,</code> <code>-l dba_home_directory</code>	<p>The home directory for the database administrator. The default is <code>/home/dbadmin</code>.</p>
<code>--dba-user-password</code> <code>dba_password,</code> <code>-p dba_password</code>	<p>The password for the database administrator account. If not supplied, the script prompts for a password and does not echo the input.</p>
<code>--dba-user-password-disabled</code>	<p>Disable the password for the <code>--dba-user</code>. This argument stops the installer from prompting for a password for the <code>--dba-user</code>. You can assign a password later using standard user management tools such as <code>passwd</code>.</p>
<code>--spread-logging,</code> <code>-w</code>	<p>Configures spread to output logging output to <code>/opt/vertica/log/spread_<hostname>.log</code>. Does not apply to upgrades.</p> <p>Note: Do not enable this logging unless directed to by Vertica Analytics Platform Technical Support.</p>
<code>--ssh-password password,</code> <code>-P password</code>	<p>The password to use by default for each cluster host. If not supplied, and the <code>-i</code> option is not used, then the script prompts for the password if and when necessary and does not echo the input. Do not use with the <code>-i</code> option.</p> <p>Special note about password:</p> <p>If you run the <code>install_vertica</code> script as root, specify the root password with the <code>-P</code> parameter:</p> <pre># /opt/vertica/sbin/install_vertica -P <root_passwd></pre>

Option	Description
	<p>If, however, you run the <code>install_vertica</code> script with the <code>sudo</code> command, the password for the <code>-P</code> parameter should be the password of the user who runs <code>install_vertica</code>, not the root password. For example if user <code>dbadmin</code> runs <code>install_vertica</code> with <code>sudo</code> and has a password with the value <code>dbapasswd</code>, then the value for <code>-P</code> should be <code>dbapasswd</code>:</p> <pre>\$ sudo /opt/vertica/sbin/install_vertica -P dbapasswd</pre>
<pre>--ssh-identity file, -i file</pre>	<p>The root private-key <i>file</i> to use if passwordless ssh has already been configured between the hosts. Verify that normal SSH works without a password before using this option. The file can be private key file (for example, <code>id_rsa</code>), or PEM file. Do not use with the <code>--ssh-password/-P</code> option.</p> <p>Vertica accepts the following:</p> <ul style="list-style-type: none"> • By providing an SSH private key which is not password protected. You cannot run the <code>install_vertica</code> script with the <code>sudo</code> command when using this method. • By providing a password-protected private key and using an SSH-Agent. Note that <code>sudo</code> typically resets environment variables when it is invoked. Specifically, the <code>SSH_AUTHSOCK</code> variable required by the SSH-Agent may be reset. Therefore, configure your system to maintain <code>SSH_AUTHSOCK</code> or invoke the <code>install_vertica</code> command using a method similar to the following: <code>sudo SSH_AUTHSOCK=\$SSH_AUTHSOCK /opt/vertica/sbin/install_vertica ...</code>
<pre>--config-file file, -z file</pre>	<p>Accepts an existing properties file created by <code>--record-config file_name</code>. This properties file contains key/value parameters that map to values in the <code>install_vertica</code> script, many</p>

Option	Description
<pre>--add-hosts host_list, -A host_list</pre>	<p>with Boolean arguments that default to false.</p> <p>A comma-separated list of hosts to add to an existing Vertica cluster.</p> <p>--add-hosts modifies an existing installation of Vertica by adding a host to the database cluster and then reconfiguring the spread. This is useful for increasing system performance or setting K-safety to one (1) or two (2).</p> <p>Notes:</p> <ul style="list-style-type: none"> If you have used the -T parameter to configure spread to use direct point-to-point communication within the existing cluster, you must use the -T parameter when you add a new host; otherwise, the new host automatically uses UDP broadcast traffic, resulting in cluster communication problems that prevent Vertica from running properly. <p>Examples:</p> <pre>--add-hosts host01 --add-hosts 192.168.233.101</pre> <ul style="list-style-type: none"> The <code>update_vertica</code> script described in Adding Nodes calls the <code>install_vertica</code> script to update the installation. You can use either the <code>install_vertica</code> or <code>update_vertica</code> script with the <code>--add-hosts</code> parameter.
<pre>--record-config file_name, -B file_name</pre>	<p>Accepts a file name, which when used in conjunction with command line options, creates a properties file that can be used with the <code>--config-file</code> parameter. This parameter creates the properties file and exits; it has no impact on installation.</p>
<pre>--clean</pre>	<p>Forcibly cleans previously stored configuration</p>

Option	Description
	<p>files. Use this parameter if you need to change the hosts that are included in your cluster. Only use this parameter when no database is defined. Cannot be used with <code>update_vertica</code>.</p>
<pre>--license { license_file CE }, -L { license_file CE }</pre>	<p>Silently and automatically deploys the license key to <code>/opt/vertica/config/share</code>. On multi-node installations, the <code>--license</code> option also applies the license to all nodes declared in the <code>--hosts host_list</code>. To activate your license, use the <code>--license</code> option with the <code>--accept-eula</code> option. If you do not use the <code>--accept-eula</code> option, you are asked to accept the EULA when you connect to your database. Once you accept the EULA, your license is activated.</p> <p>If specified with <code>CE</code>, automatically deploys the Community Edition license key, which is included in your download. You do not need to specify a license file.</p> <p>Examples:</p> <pre>--license CE --license /tmp/vlicense.dat</pre>
<pre>--remove-hosts host_list, -R host_list</pre>	<p>A comma-separated list of hosts to remove from an existing Vertica cluster.</p> <p><code>--remove-hosts</code> modifies an existing installation of Vertica by removing a host from the database cluster and then reconfiguring the spread. This is useful for removing an obsolete or over-provisioned system. For example:</p> <pre>---remove-hosts host01 -R 192.168.233.101</pre> <p>Notes:</p> <ul style="list-style-type: none"> If you used the <code>-T</code> parameter to configure spread to use direct point-to-point communication within the existing cluster, you

Option	Description
	<p>must use <code>-T</code> when you remove a host; otherwise, the hosts automatically use UDP broadcast traffic, resulting in cluster communication problems that prevents Vertica from running properly.</p> <ul style="list-style-type: none"> The <code>update_vertica</code> script described in Removing Nodes in the Administrator's Guide calls the <code>install_vertica</code> script to perform the update to the installation. You can use either the <code>install_vertica</code> or <code>update_vertica</code> script with the <code>-R</code> parameter.
<pre>--control-network { BCAST_ADDR default }, -S { BCAST_ADDR default }</pre>	<p>Takes either the value 'default' or a broadcast network IP address (<code>BCAST_ADDR</code>) to allow spread communications to be configured on a subnet that is different from other Vertica data communications. <code>--control-network</code> is also used to force a cluster-wide spread reconfiguration when changing spread related options.</p> <p>Note: The <code>--control-network</code> must match the subnet for at least some of the nodes in the database. If the provided address does not match the subnet of any node in the database then the installer displays an error and stops. If the provided address matches some, but not all of the node's subnets, then a warning is displayed, but the install continues. Ideally, the value for <code>--control-network</code> should match all node subnets.</p> <p>Examples:</p> <pre>--control-network default --control-network 10.20.100.255</pre>
<pre>--point-to-point, -T</pre>	<p>Configures spread to use direct point-to-point communication between all Vertica nodes. You</p>

Option	Description
	<p>should use this option if your nodes aren't located on the same subnet. You should also use this option for all virtual environment installations, regardless of whether the virtual servers are on the same subnet or not. The maximum number of spread daemons supported in point-to-point communication in Vertica is 80. It is possible to have more than 80 nodes by using large cluster mode, which does not install a spread daemon on each node.</p> <p>Cannot be used with <code>--broadcast</code>, as the setting must be either <code>--broadcast</code> or <code>--point-to-point</code>.</p> <p>Important: When changing the configuration from <code>--broadcast</code> (the default) to <code>--point-to-point</code> or from <code>--point-to-point</code> to <code>--broadcast</code>, the <code>--control-network</code> parameter must also be used.</p> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <p>Note: Spread always runs on UDP. <code>-T</code> does not denote TCP.</p> </div>
<p><code>--broadcast</code>, <code>-U</code></p>	<p>Specifies that Vertica use UDP broadcast traffic by spread between nodes on the subnet. This parameter is automatically used by default. No more than 80 spread daemons are supported by broadcast traffic. It is possible to have more than 80 nodes by using large cluster mode, which does not install a spread daemon on each node.</p> <p>Cannot be used with <code>--point-to-point</code>, as the setting must be either <code>--broadcast</code> or <code>--point-to-point</code>.</p> <p>Important: When changing the configuration from <code>--broadcast</code> (the default) to <code>--point-to-point</code> or from <code>--point-to-point</code> to <code>--broadcast</code>, the <code>--control-network</code> parameter must also be used.</p>

Option	Description
	<p>Note: Spread always runs on UDP. -U does not mean use UDP instead of TCP.</p>
<p>--accept-eula, -Y</p>	<p>Silently accepts the EULA agreement. On multi-node installations, the --accept-eula value is propagated throughout the cluster at the end of the installation, at the same time as the Administration Tools metadata.</p> <p>Use the --accept-eula option with the --license option to activate your license.</p>
<p>--no-system-configuration</p>	<p>By default, the installer makes system configuration changes to meet server requirements. If you do not want the installer to change any system properties, then use the --no-system-configuration. The installer presents warnings or failures for configuration settings that do not meet requirements that it normally would have automatically configured.</p> <p>Note: The system user account is still created/updated when using this parameter.</p>
<p>--failure-threshold</p>	<p>Stops the installation when the specified failure threshold is encountered.</p> <p>Options can be one of:</p> <ul style="list-style-type: none"> • HINT - Stop the install if a HINT or greater issue is encountered during the installation tests. HINT configurations are settings you should make, but the database runs with no significant negative consequences if you omit the setting. • WARN (default) - Stop the installation if a WARN or greater issue is encountered. WARN issues may affect the performance of the database. However, for basic testing purposes or Community Edition users, WARN issues can

Option	Description
	<p>be ignored if extreme performance is not required.</p> <ul style="list-style-type: none"> • FAIL - Stop the installation if a FAIL or greater issue is encountered. FAIL issues can have severely negative performance consequences and possible later processing issues if not addressed. However, Vertica can start even if FAIL issues are ignored. • HALT - Stop the installation if a HALT or greater issue is encountered. The database may not be able to be started if you choose his option. Not supported in production environments. • NONE - Do not stop the installation. The database may not start. Not supported in production environments.
<pre>--large-cluster, -2 [<integer> DEFAULT]</pre>	<p>Enables a large cluster layout, in which control message responsibilities are delegated to a subset of Vertica Analytics Platform nodes (called control nodes) to improve control message performance in large clusters. Consider using this parameter with more than 50 nodes.</p> <p>Options can be one of:</p> <ul style="list-style-type: none"> • <i><integer></i>—The number of control nodes you want in the cluster. Valid values are 1 to 120 for all new databases. • DEFAULT—Vertica Analytics Platform chooses the number of control nodes using calculations based on the total number of cluster nodes in the <code>--hosts</code> argument. <p>For more information, see Large Cluster in the Administrator's Guide.</p>

Installing Vertica Silently

This section describes how to create a properties file that lets you install and deploy Vertica-based applications quickly and without much manual intervention.

Note: The procedure assumes that you have already performed the tasks in [Before You Install Vertica](#).

Install the properties file:

1. Download and install the Vertica install package, as described in [Installing Vertica](#).
2. Create the properties file that enables non-interactive setup by supplying the parameters you want Vertica to use. For example:

The following command assumes a multi-node setup:

```
# /opt/vertica/sbin/install_vertica --record-config file_name --license /tmp/license.txt --  
accept-eula \  
# --dba-user-password password --ssh-password password --hosts host_list --rpm package_name
```

The following command assumes a single-node setup:

```
# /opt/vertica/sbin/install_vertica --record-config file_name --license /tmp/license.txt --  
accept-eula \  
# --dba-user-password password
```

Option	Description
<code>--record-file <i>file_name</i></code>	[Required] Accepts a file name, which when used in conjunction with command line options, creates a properties file that can be used with the <code>--config-file</code> option during setup. This flag creates the properties file and exits; it has no impact on installation.
<code>--license { <i>license_file</i> CE }</code>	Silently and automatically deploys the license key to <code>/opt/vertica/config/share</code> . On multi-node installations, the <code>--license</code> option also applies the license to all nodes declared in the <code>--hosts <i>host_list</i></code> . If specified with CE, automatically deploys the

Option	Description
	Community Edition license key, which is included in your download. You do not need to specify a license file.
<code>--accept-eula</code>	Silently accepts the EULA agreement during setup.
<code>--dba-user-password <i>password</i></code>	The password for the Database Administrator account; if not supplied, the script prompts for the password and does not echo the input.
<code>--ssh-password <i>password</i></code>	The root password to use by default for each cluster host; if not supplied, the script prompts for the password if and when necessary and does not echo the input.
<code>--hosts <i>host_list</i></code>	<p>A comma-separated list of hostnames or IP addresses to include in the cluster; do not include space characters in the list.</p> <p>Examples:</p> <pre>--hosts host01,host02,host03 --hosts 192.168.233.101,192.168.233.102,192.168.233.103</pre>
<code>--rpm <i>package_name</i></code> <code>--deb <i>package_name</i></code>	<p>The name of the RPM or Debian package that contained this script.</p> <p>Example:</p> <pre>--rpm vertica_8.1.x.x86_64.RHEL6.rpm</pre> <p>This parameter is required on multi-node installations if the RPM or DEB package is not already installed on the other hosts.</p>

See [Installing Vertica with the Installation Script](#) for the complete set of installation parameters.

Tip: Supply the parameters to the properties file once only. You can then install Vertica using just the `--config-file` parameter, as described below.

3. Use one of the following commands to run the installation script.

- If you are root:

```
/opt/vertica/sbin/install_vertica --config-file file_name
```

- If you are using sudo:

```
$ sudo /opt/vertica/sbin/install_vertica --config-file file_name
```

--config-file *file_name* accepts an existing properties file created by --record-config *file_name*. This properties file contains key/value parameters that map to values in the `install_vertica` script, many with boolean arguments that default to false

The command for a single-node install might look like this:

```
# /opt/vertica/sbin/install_vertica --config-file /tmp/vertica-inst.prp
```

4. If you did not supply a --ssh-password password parameter to the properties file, you are prompted to provide the requested password to allow installation of the RPM/DEB and system configuration of the other cluster nodes. If you are root, this is the root password. If you are using sudo, this is the sudo user password. The password does not echo on the command line.

Note: If you are root on a single-node installation, you are not prompted for a password.

5. If you did not supply a --dba-user-password password parameter to the properties file, you are prompted to provide the database administrator account password.

The installation script creates a new Linux user account (dbadmin by default) with the password that you provide.

6. Carefully examine any warnings produced by `install_vertica` and correct the problems if possible. For example, insufficient RAM, insufficient Network throughput and too high readahead settings on filesystem could cause performance problems later on.

Note: You can redirect any warning outputs to a separate file, instead of having them display on the system. Use your platforms standard redirected mechanisms. For

```
example: install_vertica [options] > /tmp/file 1>&2.
```

7. **Optionally** perform the following steps:

- [Install the ODBC and JDBC driver.](#)
- [Install the vsqll client application on non-cluster hosts.](#)

8. Disconnect from the Administration Host as instructed by the script. This is required to:

- Set certain system parameters correctly.
- Function as the Vertica database administrator.

At this point, Linux root privileges are no longer needed. The database administrator can perform the remaining steps.

Note: When creating a new database, the database administrator might want to use different data or catalog locations than those created by the installation script. In that case, a Linux administrator might need to create those directories and change their ownership to the database administrator.

- If you supplied the `--license` and `--accept-eula` parameters to the properties file, then proceed to the [Getting Started](#) and then see [Configuring the Database](#) in the Administrator's Guide. Otherwise:
 1. Log in to the Database Administrator account on the administration host.
 2. Accept the End User License Agreement and install the license key you downloaded previously as described in [Install the License Key](#).
 3. Proceed to [Getting Started](#) and then see [Configuring the Database](#) in the Administrator's Guide.

Notes

- Downgrade installations are not supported.
- The following is an example of the contents of the configuration properties file:

```
accept_eula = True  
license_file = /tmp/license.txt  
record_to = file_name  
root_password = password  
vertica_dba_group = verticadba  
vertica_dba_user = dbadmin  
vertica_dba_user_password = password
```

Installing Vertica on Amazon Web Services (AWS)

Beginning with Vertica 6.1.x, you can use Vertica on AWS by utilizing a pre-configured Amazon Machine Image (AMI). For details on installing and configuring a cluster on AWS, refer to [Installing and Running Vertica on AWS](#).

Installing and Configuring Management Console

This section describes how to install, configure, and upgrade Management Console (MC). If you need to back up your instance of MC, see [Backing Up MC](#) in the Administrator's Guide.

You can install MC before or after you install Vertica; however, consider installing Vertica and creating a database before you install MC.

Before You Install MC

Management Console (MC) 8.1.x is compatible with the latest hotfix version of Vertica server 7.2.3 and above. Read the following documents for more information:

- Supported Platforms document, at <http://my.vertica.com/docs>. The Supported Platforms document also lists supported browsers for MC.
- [Installation Overview and Checklist](#). Make sure you have everything ready for your Vertica configuration.
- [Before You Install Vertica](#). Read for required prerequisites for *all* Vertica configurations, including Management Console.

Driver Requirements for Linux SuSe Distributions

The MC (`vertica-console`) package contains the Oracle Implementation of Java 7 JRE and requires that you install the unixODBC driver manager on SuSe Linux platforms. unixODBC provides needed libraries `libodbc` and `libodbcinst`.

Port Requirements

When you use MC to create a Vertica cluster, the [Create Cluster Wizard](#) uses SSH on its default port (22).

Port 5444 is the default agent port and must be available for MC-to-node and node-to-node communications.

Port 5450 is the default MC port and must be available for node-to-MC communications.

See [Ensure Ports Are Available](#) for more information about port and firewall considerations.

Firewall Considerations

Make sure that a firewall or iptables are not blocking communications between the cluster's database, Management Console, and MC's agents on each cluster node.

IP Address Requirements

If you install MC on a server outside the Vertica cluster it will be monitoring, that server must be accessible to at least the public network interfaces on the cluster.

Hardware Requirements

Requirements	CPU	RAM	Disk Space
Minimum	4-core	4G	2G
Recommended	8-core	8G	2G

You can install MC on any node in the cluster, or its own dedicated node. When running the MC on a node in the cluster, note that MC shares RAM and time on CPU cores with other Vertica processes. See [Disk Space Requirements for Vertica](#).

Time Synchronization and MC's Self-Signed Certificate

When you [connect to MC](#) through a client browser, Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your Vertica cluster, and on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See [Set Up Time Synchronization](#) in [Installing Vertica](#).

SSL Requirements

The `openssl` package must be installed on your Linux environment so SSL can be set up during the MC configuration process. See [SSL Overview](#) in the Administrator's Guide.

File Permission Requirements

On your local workstation, you must have at least read/write privileges on any files you plan to upload to MC through the [Cluster Installation Wizard](#). These files include the Vertica server package, the license key (if needed), the private key file, and an optional CSV file of IP addresses.

Monitor Resolution

Management Console requires a minimum resolution of 1024 x 768, but Micro Focus recommends higher resolutions for optimal viewing.

Installing Management Console

You can install Management Console on any node you plan to include in the Vertica database cluster, as well as on its own, dedicated server outside the cluster.

Install Management Console on the MC Server

1. Download the MC package from the [myVertica](#) portal:

```
vertica-console-current-version.Linux-distro)
```

Save the package to a location on the target server, such as /tmp.

2. On the target server, log in as root or a user with sudo privileges.
3. Change to the directory where you saved the MC package.
4. Install MC using your local Linux distribution package management system—rpm, yum, zypper, apt, dpkg. For example:

Red Hat 6

```
# rpm -Uvh vertica-console-current-version.x86_64.RHEL6.rpm
```

Debian and Ubuntu

```
# dpkg -i vertica-console-current-version.deb
```

5. If you stopped the database before upgrading MC, restart the database.

As the root user, use the following command:

```
/etc/init.d/verticad start
```

For versions of Red Hat 7/CentOS 7 and above, run:

```
# systemctl start vertica-consoled
```

6. Open a browser and enter the URL of the MC installation, one of the following:

- IP address:

```
https://ip-address:mc-port/
```

- Server host name:

```
https://hostname:mc-port/
```

By default, *mc-port* is 5450.

7. If MC was not previously configured, the Configuration Wizard dialog box appears. Configuration steps are described in [Configuring MC](#).

If MC was previous configured, Vertica prompts you to accept the end-user license agreement (EULA) when you first log in to MC after the upgrade.

Configuring MC

After you [install MC](#), you need to configure it through a client browser connection. An MC configuration wizard walks you through creating the Linux MC super administrator account, storage locations, and other settings that MC needs to run. Information you provide during the configuration process is stored in the `/opt/vconsole/config/console.properties` file.

If you need to change settings after the configuration wizard ends, such as port assignments, you can do so later through Home > MC Settings page.

How to Configure MC

1. Open a browser session.
2. Enter the IP address or host name of the server on which you installed MC (or any cluster node's IP/host name if you already installed Vertica), and include the default MC port 5450. For example, you'll enter one of:

```
https://xx.xx.xx.xx:5450/ https://hostname:5450/
```

3. Follow the configuration wizard.

About Authentication for the MC Super Administrator

In the final step of the configuration process, you choose an authentication method for the MC super administrator. You can decide to have MC authenticate the MC super (in which case the process is complete), or you can choose LDAP.

If you choose LDAP, provide the following information for the newly-created MC super administrator:

- Corporate LDAP service host (IP address or host name)
- LDAP server running port (default 389)
- LDAP DN (distinguished name) for base search/lookup/authentication criteria

At a minimum, specify the dc (domain component) field. For example: `dc=vertica, dc=com` generates a unique identifier of the organization, like the corporate Web URL `vertica.com`

- Default search path for the organization unit (ou)

For example: `ou=sales, ou=engineering`

- Search attribute for the user name (uid), common name (cn), and so on

For example, `uid=jdoe, cn=Jane Doe`

- Binding DN and password for the MC super administrator.

In most cases, you provide the "Bind as administrator" fields, information used to establish the LDAP service connection for all LDAP operations, like search. Instead of using the administrator user name and password, the MC administrator could use his or her own LDAP credentials, as long as that user has search privileges.

If You Choose Bind Anonymously

Unless you specifically configure the LDAP server to deny anonymous binds, the underlying LDAP protocol will not cause MC's [Configure Authentication](#) process to fail if you choose "Bind anonymously" for the MC administrator. Before you use anonymous bindings for LDAP authentication on MC, be sure that your LDAP server is configured to explicitly disable/enable this option. For more information, see the article on [Infusion Technology Solutions](#) and the [OpenLDAP documentation](#) on access control.

What Happens Next

Shortly after you click Finish, you should see a status in the browser; however, for several seconds you might see only an empty page. During this brief period, MC runs as the local user 'root' long enough to bind to port number 5450. Then MC switches to the MC super administrator account that you just created, restarts MC, and displays the MC login page.

Where to Go Next

If you are a new MC user and this is your first MC installation, you might want to familiarize yourself with MC design. See [Management Console](#) in Vertica Concepts.

If you'd rather use MC now, the following following topics in the Administrator's Guide should help get you started:

If you want to ...	See ...
Use the MC interface to install Vertica on a cluster of hosts	Creating a Cluster Using MC
Create a new, empty Vertica database or import an existing Vertica database cluster into the MC interface	Managing Database Clusters
Create new MC users and map them to one or more Vertica databases that you manage through the MC interface	Managing Users and Privileges (About MC Users and About MC Privileges and Roles)
Monitor MC and one or more MC-managed Vertica databases	Monitoring Vertica Using Management Console
Change default port assignments or upload a new Vertica license or SSL certificate	Managing MC Settings
Compare MC functionality to functionality that the Administration Tools provides	Administration Tools and Management Console

Creating a Cluster Using MC

You can use Management Console to install a Vertica cluster on hosts where Vertica software has not been installed. The Cluster Installation wizard lets you specify the hosts you want to include in your Vertica cluster, loads the Vertica software onto the hosts, validates the hosts, and assembles the nodes into a cluster.

Management Console must be installed and configured before you can create a cluster on targeted hosts. See [Installing and Configuring the MC](#) for details.

Steps Required to Install a Vertica Cluster Using MC:

- [Install and configure MC](#)
- [Prepare the Hosts](#)
- [Create the private key file](#) and copy it to your local machine
- [Run the Cluster Installation Wizard](#)
- [Validate the hosts and create the cluster](#)
- [Create a new database on the cluster](#)

Prepare the Hosts

Before you can install a Vertica cluster using the MC, you must prepare each host that will become a node in the cluster. The cluster creation process runs validation tests against each host before it attempts to install the Vertica software. These tests ensure that the host is correctly configured to run Vertica.

Install Perl

The MC cluster installer uses Perl to perform the installation. Install Perl 5 on the target hosts before performing the cluster installation. Perl is available for download from www.perl.org.

Validate the Hosts

The validation tests provide:

- Warnings and error messages when they detect a configuration setting that conflicts with the Vertica requirements or any performance issue
- Suggestions for configuration changes when they detect an issue

Note: The validation tests do not automatically fix all problems they encounter.

All hosts must pass validation before the cluster can be created.

If you accepted the default configuration options when installing the OS on your host, then the validation tests will likely return errors, since some of the default options used on Linux systems conflict with Vertica requirements. See [Installing Vertica](#) for details on OS settings. To speed up the validation process you can perform the following steps on the prospective hosts before you attempt to validate the hosts. These steps are based on Red Hat Enterprise Linux and CentOS systems, but other supported platforms have similar settings.

On each host you want to include in the Vertica cluster, you must stage the host according to [Before You Install Vertica](#).

Create a Private Key File

Before you can install a cluster, Management Console must be able to access the hosts on which you plan to install Vertica. MC uses password-less SSH to connect to the hosts and install Vertica software using a private key file.

If you already have a private key file that allows access to all hosts in the potential cluster, you can use it in the cluster creation wizard.

Note: The private key file is required to complete the MC cluster installation wizard.

Create a Private Key File

1. Log into the server as root or as a user with sudo privileges.
2. Change to your home directory.

```
$ cd ~
```

3. Create an `.ssh` directory if one does not already exist.

```
$ mkdir .ssh
```

4. Generate a passwordless private key/public key pair.

```
$ ssh-keygen -q -t rsa -f ~/.ssh/vid_rsa -N ''
```

This command creates two files: `vid_rsa` and `vid_rsa.pub`. The `vid_rsa` file is the private key file that you upload to the MC so that it can access nodes on the cluster and install Vertica. The `vid_rsa.pub` file is copied to all other hosts so that they can be accessed by clients using the `vid_rsa` file.

5. Make your `.ssh` directory readable and writable only by yourself.

```
$ chmod 700 /root/.ssh
```

6. Change to the `.ssh` directory.

```
$ cd ~/.ssh
```

7. Edit `sshd.config` as follows to disable password authentication for root:

```
PermitRootLogin without-password
```

8. Concatenate the public key into to the file `vauthorized_keys2`.

```
$ cat vid_rsa.pub >> vauthorized_keys2
```

9. If the host from which you are creating the public key will also be in the cluster, copy the public key into the local-hosts authorized key file:

```
cat vid_rsa.pub >> authorized_keys
```

10. Make the files in your `.ssh` directory readable and writable only by yourself.

```
$ chmod 600 ~/.ssh/*
```

11. Create the `.ssh` directory on the other nodes.

```
$ ssh <host> "mkdir /root/.ssh"
```

12. Copy the vauthorized key file to the other nodes.

```
$ scp -r /root/.ssh/vauthorized_keys2 <host>:/root/.ssh/.
```

13. On each node, concatenate the `vauthorized_keys2` public key to the `authorized_keys` file and make the file readable and writable only by the owner.

```
$ ssh <host> "cd /root/.ssh;;cat vauthorized_keys2 >> authorized_keys; chmod 600 /root/.ssh/authorized_keys"
```

14. On each node, remove the `vauthorized_keys2` file.

```
$ ssh -i /root/.ssh/vid_rsa <host> "rm /root/.ssh/vauthorized_keys2"
```

15. Copy the `vid_rsa` file to the workstation from which you will access the MC cluster installation wizard. This file is required to install a cluster from the MC.

A complete example of the commands for creating the public key and allowing access to three hosts from the key is below. The commands are being initiated from the `docg01` host, and all hosts will be included in the cluster (`docg01` - `docg03`):

```
ssh docg01
cd ~/.ssh
ssh-keygen -q -t rsa -f ~/.ssh/vid_rsa -N ''
cat vid_rsa.pub > vauthorized_keys2
cat vid_rsa.pub >> authorized_keys
chmod 600 ~/.ssh/*
scp -r /root/.ssh/vauthorized_keys2 docg02:/root/.ssh/.
scp -r /root/.ssh/vauthorized_keys2 docg03:/root/.ssh/.
ssh docg02 "cd /root/.ssh;;cat vauthorized_keys2 >> authorized_keys; chmod 600 /root/.ssh/authorized_keys"
ssh docg03 "cd /root/.ssh;;cat vauthorized_keys2 >> authorized_keys; chmod 600 /root/.ssh/authorized_keys"
ssh -i /root/.ssh/vid_rsa docg02 "rm /root/.ssh/vauthorized_keys2"
ssh -i /root/.ssh/vid_rsa docg03 "rm /root/.ssh/vauthorized_keys2"
rm ~/.ssh/vauthorized_keys2
```

Use the MC Cluster Installation Wizard

The Cluster Installation Wizard guides you through the steps required to install a Vertica cluster on hosts that do not already have Vertica software installed.

Note: If you are using MC with the Vertica AMI on Amazon Web Services, note that the Create Cluster and Import Cluster options are not supported.

Prerequisites

Before you proceed, make sure you:

- [Installed and configured MC](#).
- [Prepared the hosts](#) that you will include in the Vertica database cluster.
- [Created the private key \(pem\) file](#) and copied it to your local machine.
- Obtained a copy of your Vertica license if you are installing the Premium Edition. If you are using the Community Edition, a license key is not required.
- Downloaded the Vertica server RPM (or DEB file).
- Have read/copy permissions on files stored on the local browser host that you will transfer to the host on which MC is installed.

Permissions on Files to Transfer to MC

On your local workstation, you must have at least read/write privileges on files you'll upload to MC through the Cluster Installation Wizard. These files include the Vertica server package, the license key (if needed), the private key file, and an optional CSV file of IP addresses.

Create a New Vertica Cluster Using MC

1. [Connect](#) to Management Console and log in as an MC administrator.
2. On MC's [Home page](#), click the **Provisioning** task. The Provisioning dialog appears.
3. Click **Create a new cluster**.
4. The Create Cluster wizard opens. Provide the following information:

- a. Cluster name—A label for the cluster
- b. Vertica Admin User—The user that is created on each of the nodes when they are installed, typically 'dbadmin'. This user has access to Vertica and is also an OS user on the host.
- c. Password for the Vertica Admin User—The password you enter (required) is set for each node when MC installs Vertica.

Note: MC does not support an empty password for the administrative user.

- d. Vertica Admin Path—Storage location for catalog files, which defaults to /home/dbadmin unless you specified a different path during MC configuration (or later on MC's Settings page).

Important: The Vertica Admin Path must be the same as the Linux database administrator's home directory. If you specify a path that is not the Linux dbadmin's home directory, MC returns an error.

5. Click **Next** and specify the private key file and host information:

- a. Click **Browse** and navigate to the private key file (vid_rsa) that you created earlier.

Note: You can change the private key file at the beginning of the validation stage by clicking the name of the private key file in the bottom-left corner of the page. However, you cannot change the private key file after validation has begun unless the first host fails validation due to an SSH login error.

- b. Include the host IP addresses. You have three options:

Specify later (but include number of nodes). This option allows you to specify the number of nodes, but not the specific IPs. You can specify the specific IPs before you validate hosts.

Import IP addresses from local file. You can specify the hosts in a CSV file using either IP addresses or host names.

Enter a range of IP addresses. You can specify a range of IPs to use for new nodes. For example 192.168.1.10 to 192.168.1.30. The range of IPs must be on the same or contiguous subnets.

6. Click **Next** and select the software and license:
 - a. Vertica Software. If one or more Vertica packages have been uploaded, you can select one from the list. Otherwise, select **Upload a new local vertica binary file** and browse to a Vertica server file on your local system.
 - b. Vertica License. Click **Browse** and navigate to a local copy of your Vertica license if you are installing the Premium Edition. Community Edition versions require no license key.
7. Click **Next**. The Create cluster page opens. If you did not specify the IP addresses, select each host icon and provide an IP address by entering the IP in the box and clicking **Apply** for each host you add.

You are now ready to [Validate Hosts and Create the Cluster](#).

Validate Hosts and Create the Cluster

Host validation is the process where the MC runs tests against each host in a [proposed cluster](#).

You can validate hosts only after you have completed the cluster installation wizard. You must validate hosts before the MC can install Vertica on each host.

At any time during the validation process, but before you create the cluster, you can add and remove hosts by clicking the appropriate button in the upper left corner of the page on MC. A Create Cluster button appears when all hosts that appear in the node list are validated.

How to Validate Hosts

To validate one or more hosts:

1. [Connect](#) to Management Console and log in as an MC administrator.
2. On the MC [Home page](#), click the **Databases and Clusters** task.
3. In the list of databases and clusters, select the cluster on which you have recently run the cluster installation wizard (**Creating...** appears under the cluster) and click **View**.
4. Validate one or several hosts:
 - To validate a single host, click the host icon, then click **Validate Host**.
 - To validate all hosts at the same time, click **All** in the Node List, then click **Validate Host**.

- To validate more than one host, but not all of them, Ctrl+click the host numbers in the node list, then click **Validate Host**.

5. Wait while validation proceeds.

The validation step takes several minutes to complete. The tests run in parallel for each host, so the number of hosts does not necessarily increase the amount of time it takes to validate all the hosts if you validate them at the same time. Hosts validation results in one of three possible states:

- Green check mark—The host is valid and can be included in the cluster.
- Orange triangle—The host can be added to the cluster, but warnings were generated. Click the tests in the host validation window to see details about the warnings.
- Red X—The host is not valid. Click the tests in the host validation window that have red X's to see details about the errors. You must correct the errors re-validate or remove the host before MC can create the cluster.

To remove an invalid host: Highlight the host icon or the IP address in the Node List and click **Remove Host**.

All hosts must be valid before you can create the cluster. Once all hosts are valid, a **Create Cluster** button appears near the top right corner of the page.

How to Create the Cluster

1. Click **Create Cluster** to install Vertica on each host and assemble the nodes into a cluster.

The process, done in parallel, takes a few minutes as the software is copied to each host and installed.

2. Wait for the process to complete. When the **Success** dialog opens, you can do one of the following:

- Optionally create a database on the new cluster at this time by clicking **Create Database**
- Click **Done** to create the database at a later time

See [Creating a Database on a Cluster](#) for details on creating a database on the new cluster.

Create a Database on a Cluster

After you use the [MC Cluster Installation Wizard](#) to create a Vertica cluster, you can create a database on that cluster through the MC interface. You can create the database on all cluster nodes or on a subset of nodes.

If a database had been created using the Administration Tools on any of the nodes, MC detects (autodiscovers) that database and displays it on the Manage (Cluster Administration) page so you can import it into the MC interface and begin monitoring it.

MC allows only one database running on a cluster at a time, so you might need to stop a running database before you can create a new one.

The following procedure describes how to create a database on a cluster that you created using the MC [Cluster Installation Wizard](#). To create a database on a cluster that you created by running the `install_vertica` script, see [Creating an Empty Database](#).

Create a Database on a Cluster

To create a new empty database on a new cluster:

1. If you are already on the **Databases and Clusters** page, skip to the next step. Otherwise:
 - a. [Connect](#) to MC and sign in as an MC administrator.
 - b. On the [Home page](#), click **Existing Infrastructure**.
2. If no databases exist on the cluster, continue to the next step. Otherwise:
 - a. If a database is running on the cluster on which you want to add a new database, select the database and click **Stop**.
 - b. Wait for the running database to have a status of *Stopped*.
3. Click the cluster on which you want to create the new database and click **Create Database**.
4. The Create Database wizard opens. Provide the following information:
 - Database name and password. See [Creating a Database Name and Password](#) for rules.
 - Optionally click **Advanced** to open the advanced settings and change the port, and catalog path, and data path. By default the MC application/web server port is 5450 and

paths are `/home/dbadmin`, or whatever you defined for the paths when you ran the cluster creation wizard. Do not use the default agent port 5444 as a new setting for the MC application/web server port. See **MC Settings > Configuration** for port values.

5. Click **Continue**.

6. Select nodes to include in the database.

The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node and clear **Include host in new database** to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the **Include** check box.

7. Click **Create** in the Database Configuration window to create the database on the nodes.

The creation process takes a few moments and then the database is started and a **Success** message appears.

8. Click **OK** to close the success message.

The Database Manager page opens and displays the database nodes. Nodes not included in the database are gray.

After You Install Vertica

The tasks described in this section are optional and are provided for your convenience. When you have completed this section, proceed to one of the following:

- [Using This Guide](#) in Getting Started
- [Configuring the Database](#) in the Administrator's Guide

Install the License Key

If you did not supply the `-L` parameter during setup, or if you did not bypass the `-L` parameter for a [silent install](#), the first time you log in as the Database Administrator and run the Vertica Administration Tools or Management Console, Vertica requires you to install a license key.

Follow the instructions in [Managing Licenses](#) in the Administrator's Guide.

Optionally Install vsql Client Application on Non-Cluster Hosts

You can use the Vertica `vsql` executable image on a non-cluster Linux host to connect to a Vertica database.

- On Red Hat, CentOS, and SUSE systems, you can install the client driver RPM, which includes the `vsql` executable. See [Installing the Client RPM on Red Hat and SUSE](#) for details.
- If the non-cluster host is running the same version of Linux as the cluster, copy the image file to the remote system. For example:

```
$ scp host01:/opt/vertica/bin/vsql . $ ./vsql
```

- If the non-cluster host is running a different version of Linux than your cluster hosts, and that operating system is not Red Hat version 5 64-bit or SUSE 10/11 64-bit, you must install the Vertica server RPM in order to get `vsql`. Download the appropriate rpm package from the Download tab of the [myVertica portal](#) then log into the non-cluster host as root and install the rpm package using the command:

```
# rpm -Uvh filename
```

In the above command, *filename* is the package you downloaded. Note that you do not have to run the `install_Vertica` script on the non-cluster host in order to use `vsql`.

Notes

- Use the same [Command-Line Options](#) that you would on a cluster host.
- You cannot run `vsql` on a Cygwin bash shell (Windows). Use `ssh` to connect to a cluster host, then run `vsql`.

`vsql` is also available for additional platforms. See [Installing the vsql Client](#).

Installing Client Drivers

After you install Vertica, install drivers on the client systems from which you plan to access your databases. Micro Focus supplies drivers for ADO.NET, JDBC, ODBC, OLE DB, Perl, and Python. For instructions on installing these drivers, see [Client Drivers](#) in [Connecting to Vertica](#).

Creating a Database

To get started using Vertica immediately after installation, create a database. You can use either the Administration Tools or the Management Console.

Creating a Database Using the Administration Tools

Follow these step to create a database using the Administration Tools.

1. Log in as the database administrator, and type `admintools` to bring up the Administration Tools.
2. When the EULA (end-user license agreement) window opens, type `accept` to proceed. A window displays, requesting the location of the license key file you downloaded from the Micro Focus Web site. The default path is `/tmp/vlicense.dat`.

- If you are using the Vertica Community Edition, click **OK** without entering a license key.
 - If you are using the Vertica Premium Edition, type the absolute path to your license key (for example, `/tmp/vlicense.dat`) and click **OK**.
3. From the Administration Tools **Main Menu**, click **Configuration Menu**, and then click **OK**.
 4. Click **Create Database**, and click **OK** to start the database creation wizard.

To create a database using MC, refer [Creating a Database Using MC](#).

See Also

- [Using the Vertica Interfaces](#)

Upgrading Vertica

The process of upgrading your database with a new Vertica version includes:

- [Complete upgrade prerequisites](#)
- [Upgrade Vertica](#)
- [Perform post-upgrade tasks—required, recommended, and optional](#)

Click on the above links for detailed instructions.

Upgrade Paths

Upgrades are incremental. To upgrade successfully, use the following paths:

- Vertica 7.0 to 7.1
- Vertica 7.1 to 7.2
- Vertica 7.2 to 8.0
- Vertica 8.0 to 8.1

Note: You cannot upgrade Vertica from versions 7.x to a FIPS-enabled 8.0 system. For more information on FIPS see [Federal Information Processing Standard](#).

Be sure to read the Release Notes and New Features for each version in your path. Documentation for the current Vertica version is available in the RPM, and at <http://my.vertica.com/docs>, which also provides access to documentation for earlier versions.

To upgrade from an earlier version, please contact [Vertica Technical Support](#) for assistance.

Before You Upgrade

Before you upgrade the Vertica database, you must perform the following steps:

- [Perform a full database backup](#). This precautionary measure allows you to restore the current version, if the upgrade is unsuccessful.
- [Verify platform requirements](#) for the new version.
- [Check catalog storage space](#).
- [Uninstall the HDFS connector](#) (required only if you are upgrading from pre-7.2 versions).
- [Back up any Geospatial indexes](#) and save the results in a temporary table.

After you complete these tasks, shut down the database gracefully. This procedure is described in the Administrator's Guide: [Stopping the Database](#).

Verifying Platform Requirements

The Vertica installer checks the target platform as it runs, and stops whenever it determines the platform fails to meet an installation requirement. Before you update the server package on your systems, manually verify that your platform meets all hardware and software requirements (see [Platform Requirements and Recommendations](#)).

By default, the installer stops on all warnings. You can configure the level where the installer stops installation, through the installation parameter `--failure-threshold`. If you set the failure threshold to `FAIL`, the installer ignores warnings and stops only on failures.

Caution: Changing the failure threshold lets you immediately upgrade and bring up the Vertica database. However, Vertica cannot fully optimize performance until you correct all warnings.

Checking Catalog Storage Space

Compare how much space the catalog currently uses against space that is available in the same directory:

1. Use the `du` command to determine how much space the catalog directory now uses:

```
$ du -s -BG v_vmart_node0001_catalog
2G      v_vmart_node0001_catalog
```

2. Determine how much space is available in the same directory:

```
$ df -BG v_vmart_node0001_catalog
Filesystem      1G-blocks  Used Available Use% Mounted on
/dev/sda2              48G    19G    26G  43% /
```

Uninstalling HDFS Connector

Important: Perform this task only if you are upgrading from a pre-7.2 version.

As of version 7.2, the HDFS Connector is installed automatically. If you previously downloaded and installed this connector, uninstall it before you upgrade to this release of Vertica.

Back Up Geospatial Indexes

When you upgrade, the geospatial indexes are invalid. You must back up spatial indexes that contain polygon geometry data into a temporary table and rebuild them in the newer Place version using the temporary table as the set of input polygons.

The following steps backup your spatial indexes:

1. Build an index with [STV_Create_Index](#):

```
=> SELECT STV_Create_Index(123, ST_GeomFromText('POLYGON((1 2, 2 3, 3 1, 1 2))')
      USING PARAMETERS index='pol_idx') OVER();
 type  | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
 GEOMETRY | 1        | 0    | 1     | 1     | 3     | 3     |
(1 row)
```

2. Save the index polygons in a temporary table. Use [STV_Describe_Index](#) with the `list_polygons` option to get the polygons from the index. The `\d` command describes *temp-table*:

```
=> CREATE TABLE temp-table
      AS SELECT STV_Describe_Index(USING PARAMETERS index='pol_idx', list_polygons=true) OVER();

=> \d temp-table
                                List of Fields by Tables
 Schema | Table      | Column | Type      | Size | Default | Not Null | Primary Key | Foreign Key
-----+-----+-----+-----+-----+-----+-----+-----+-----
-----
```

```
public | temp-table | gid      | int      | 8      |      | f      | f      |      |
public | temp-table | state  | varchar(20) | 20     |      | f      | f      |      |
public | temp-table | geometry | geometry(141) | 141    |      | f      | f      |      |
(3 rows)
```

3. The temporary table contains a copy of all polygons and identifiers in the indexes. The following command shows the contents of the temporary table:

```
=> SELECT gid, state, ST_AsText(geometry) FROM temp-table;
gid | state  | ST_AsText
-----+-----+-----
123 | INDEXED | POLYGON ((1 2, 2 3, 3 1, 1 2))
(1 row)
```

After you upgrade, use the polygons table as input to rebuild the index. For details, see [Rebuild Geospatial Indexes](#).

Upgrade Vertica

Important: Before running the upgrade script, be sure to review the tasks described in [Before You Upgrade](#).

Repeat this procedure for each version in your [upgrade path](#):

1. Perform a full [full hard-link local backup](#) of your existing database. This precautionary measure lets you restore from the backup, if the upgrade is unsuccessful. If the upgrade fails, you can reinstall the previous version of Vertica and [restore your database](#) to that version.

If your upgrade path includes multiple versions, create a full backup with the first upgrade. For each subsequent upgrade, you can perform incremental backups. However, Micro Focus recommends full backups before each upgrade if disk space and time allow.

2. Use admintools to [stop the database](#).
3. On each host where an additional package is installed, such as the [R language pack](#), uninstall it. For example:

```
rpm -e vertica-R-lang
```

Important: If you omit this step and do not uninstall additional packages, the Vertica server package fails to install in the next step.

4. Make sure you are logged in as root or sudo and use one of the following commands to run the RPM package installer:

- If you are root and installing an RPM:

```
# rpm -Uvh pathname
```

- If you are using sudo and installing an RPM:

```
$ sudo rpm -Uvh pathname
```

- If you are using Debian:

```
$ sudo dpkg -i pathname
```

5. On the same node on which you just installed the RPM, run `update_vertica` as root or sudo. This installs the RPM on all the hosts in the cluster. For example:

Red Hat or CentOS

```
# /opt/vertica/sbin/update_vertica --rpm /home/dbadmin/vertica_8.1.x.x86_64.RHEL6.rpm --dba-user mydba
```

Debian

```
# /opt/vertica/sbin/update_vertica --deb /home/dbadmin/vertica-amd64.deb --dba-user mydba
```

The following requirements and restrictions apply:

- The DBADMIN user must be able to read the RPM or DEB file when upgrading. Some upgrade scripts are run as the DBADMIN user, and that user must be able to read the RPM or DEB file.
- Use the same options that you used when you last installed or upgraded the database. You can find these options in `/opt/vertica/config/admintools.conf`, on the `install_opts` line. For details on all options, see [Installing Vertica with the Installation Script](#).

Caution: If you omit any previous options, their default settings are restored. If you do so, or if you change any options, the upgrade script uses the new settings to reconfigure the cluster. This can cause issues with the upgraded database.

- Omit the `--hosts/-s host-list` parameter. The upgrade script automatically identifies cluster hosts.
 - If the root user is not in `/etc/sudoers`, an error appears. The installer reports this issue with **S0311**. See the [Sudoers Manual](#) for more information.
6. [Start the database](#). The start-up scripts analyze the database and perform necessary data and catalog updates for the new version.

If Vertica issues a warning stating that one or more packages cannot be installed, run the `admintools --force-reinstall` option to force reinstallation of the packages. For details, see [Reinstalling Packages](#).

7. Perform another database backup.

Post-Upgrade Tasks

After you complete the upgrade, review post-upgrade tasks in [After You Upgrade](#).

After You Upgrade

After you finish upgrading the Vertica server package on your cluster, a number of tasks remain.

Required Tasks

- [Upgrade client authentication](#) (required only if you are upgrading from pre-7.1 versions).
- Reinstall packages such as the [R language pack](#) that you uninstalled before upgrading. For each package, see its install/upgrade instructions. For R, see [Installing/Upgrading the R Language Pack for Vertica](#).

Note: Vertica Place is automatically reinstalled with the Vertica server package.

- If the upgrade was unable to install one or more packages, [reinstall them with admintools](#).
- [Upgrade Management Console](#).
- If your Vertica installation is integrated with Hadoop, [upgrade the HCatalog connector](#).

Recommended Tasks

Vertica strongly recommends that you [bundle data files](#) after completing the database upgrade.

Optional Tasks

- [Convert backups](#) so they are compatible with the new version. Database backups from pre-7.2 versions of Vertica are incompatible with the current version.
- Import directed queries that you exported from the previous version. For details, see [Batch Query Plan Export](#) and [Exporting Directed Queries from the Catalog](#).
- [Rebuild the geospatial indexes](#). For details about the pre-upgrade steps, see [Back Up Geospatial Indexes](#).

Upgrading Client Authentication

Important: Perform this task only if you are upgrading from a pre-7.1 version.

Vertica 7.1.0 changed the storage location for the client authentication records from the `vertica.conf` file to the database catalog. When you upgrade from an earlier version, client authentication records in the `vertica.conf` file are converted and inserted into the database catalog. Vertica updates the catalog information on all nodes in the cluster.

Authentication is not enabled after upgrading. As a result, all users can connect to the database. However, if they have a password, they must enter it.

After upgrading, perform the following steps to make sure that client authentication is configured correctly and enabled for use with a running database:

1. Review the client authentication methods that Vertica created during the upgrade by querying the following system tables:

System table	Contains information about...
CLIENT_AUTH	Client authentication methods that Vertica created for your database during the upgrade.
CLIENT_AUTH_PARAMS	Parameters that Vertica defined for GSS, Ident, and LDAP authentication methods.
USER_CLIENT_AUTH	An authentication method that you associate with a specific database user through the GRANT (Authentication) statement.

2. Review the `vertica.log` file to see which authentication records Vertica was not able to create during the upgrade.
3. Create required records as needed with [CREATE AUTHENTICATION](#).
4. After the upgrade, enable all defined authentication methods. For each method, enter an [ALTER AUTHENTICATION](#) statement:

```
=> ALTER AUTHENTICATION auth-method-name ENABLE;
```

5. If you use LDAP over SSL/TLS, you must define the new parameters:

- `tls_reqcert`
- `tls_cacert`

To do so, use [ALTER AUTHENTICATION](#) as follows:

```
=> ALTER AUTHENTICATION Ldap1 SET host='ldaps://abc.dc.com', binddn_prefix='CN=',  
binddn_suffix=',OU=Unit2,DC=dc,DC=com', basedn='dc=DC,dc=com',  
tls_cacert='/home/dc.com.ca.cer', tls_reqcert='never';
```

6. Create an authentication method (`LOCAL TRUST` or `LOCAL PASSWORD`) with a very high priority such as 10,000. Grant this method to the `DBADMIN` user, and set the priority using [ALTER AUTHENTICATION](#). For example:

```
=> CREATE AUTHENTICATION dbadmin_default TRUST LOCAL;  
=> ALTER AUTHENTICATION dbadmin_default PRIORITY 10000;
```

With its high priority, this new authentication method supersedes any authentication methods you create for PUBLIC. Even if you make changes to PUBLIC authentication methods, the DBADMIN user can connect to the database at any time.

Reinstalling Packages

In most cases, Vertica automatically reinstalls all default packages when you restart your database for the first time after running the upgrade script. Occasionally, however, one or more packages might fail to reinstall correctly.

To verify that Vertica succeeded in reinstalling all packages:

1. Restart the database after upgrading.
2. Enter an incorrect password.

If any packages failed to reinstall, Vertica issues a message that specifies the uninstalled packages. In this case, run the `admintools` command `install_package` with the option `--force-reinstall`:

```
$ admintools -t install_package -d db-name -p password -P pkg-spec --force-reinstall
```

Options

Option	Function
<code>-d <i>db-name</i></code> <code>--dbname=<i>db-name</i></code>	Database name
<code>-p <i>password</i></code> <code>--password=<i>password</i></code>	Database administrator password
<code>-P <i>pkg</i></code> <code>--package=<i>pkg-spec</i></code>	Specifies which packages to install, where <i>pkg</i> is one of the following: <ul style="list-style-type: none">• The name of a package—for example, <code>flextable</code>• <code>all</code>: All available packages• <code>default</code> : All default packages that are currently installed
<code>--force-reinstall</code>	Force installation of a package even if it is already installed.

Option	Function

Examples

Force reinstallation of default packages:

```
$ admintools -t install_package -d VMart -p 'password' -P default --force-reinstall
```

Force reinstallation of one package, flextable:

```
$ admintools -t install_package -d VMart -p 'password' -P flextable --force-reinstall
```

Upgrading the Advanced Analytics Package 7.2.x to Machine Learning 8.0.x

If you installed the Advanced Analytics package on a 7.2.x Vertica cluster, then you will need to run the following upgrade script to ensure backward comparability with the 8.1.x machine learning functions.

The following command should be run as the dbadmin user:

```
$ vsql -f /opt/vertica/packages/MachineLearning/ddl/install_backward_compatibility.sql
```

Upgrading Management Console

Before You Upgrade

1. Log in as root or a user with sudo privileges on the server where MC is already installed.
2. Open a terminal window and shut down the MC process:

```
# /etc/init.d/vertica-console stop
```

For versions of Red Hat 7/CentOS 7 and above, use:

```
# systemctl stop vertica-console
```

3. Back up MC to preserve configuration metadata.

Important: A full backup is required in order to restore MC to its previous state. Restoring MC is essential if the upgrade fails, or you decide to revert to the previous version of Vertica. For details, see [Backing Up MC](#).

4. Stop the database if the following conditions are true:
 - You are upgrading MC on a Vertica host where Vertica version 7.1.2-5 or below is installed.
 - MC was installed on an Ubuntu or Debian platform.

Tip: If you upgrade from Vertica 7.2.0 or later, the configuration file `/opt/vconsole/config/console.properties` retains its previous settings. These include the setting for `messageCenter.maxEntries`, which controls the number of displayed messages. To improve performance, consider setting `messageCenter.maxEntries` to a value less than 1000.

Extended Monitoring Upgrade Recommendations

If you use [Extended Monitoring](#) to monitor a database with MC, Micro Focus recommends the following upgrade procedure to avoid data loss.

1. Log in to MC as an administrator.
2. To stop the monitored database, navigate to the Existing Infrastructure > Databases and Clusters page, select the monitored database and click **Stop**.
3. On MC Settings > MC Storage DB Setup, click **Disable Streaming** to stop the storage database's collection of monitoring data.
4. To stop the storage database, navigate to the Existing Infrastructure > Databases and Clusters page, select the monitored database and click **Stop**.
5. Upgrade MC and Vertica according to Upgrade MC and [Upgrading Vertica](#) instructions.
6. To start the storage database, navigate to the Existing Infrastructure > Databases and Clusters page, select the monitored database and click **Start**.
7. Start the monitored database.

8. On MC Settings > MC Storage DB Setup, click **Enable Streaming** to enable collection of monitoring data.

To avoid data loss, enable streaming soon after starting your monitored database. While your storage database is down and streaming is disabled, the Kafka server can retain data from your running monitored database for a limited amount of time. Data loss occurs when the data exceeds the Kafka retention policy's log size or retention time limits.

Upgrade MC

1. Download the MC package from the [myVertica](#) portal:
`vertica-console-current-version.Linux-distro)`
Save the package to a location on the target server, such as `/tmp`.
2. On the target server, log in as root or a user with sudo privileges.
3. Change to the directory where you saved the MC package.
4. Install MC using your local Linux distribution package management system—rpm, yum, zypper, apt, dpkg. For example:

Red Hat 6

```
# rpm -Uvh vertica-console-current-version.x86_64.RHEL6.rpm
```

Debian and Ubuntu

```
# dpkg -i vertica-console-current-version.deb
```

5. If you stopped the database before upgrading MC, restart the database.

As the root user, use the following command:

```
/etc/init.d/verticad start
```

For versions of Red Hat 7/CentOS 7 and above, run:

```
# systemctl start vertica-consoled
```

6. Open a browser and enter the URL of the MC installation, one of the following:

- IP address:

```
https://ip-address:mc-port/
```

- Server host name:

```
https://hostname:mc-port/
```

By default, *mc-port* is 5450.

7. If MC was not previously configured, the Configuration Wizard dialog box appears. Configuration steps are described in [Configuring MC](#).

If MC was previous configured, Vertica prompts you to accept the end-user license agreement (EULA) when you first log in to MC after the upgrade.

Recommended Post-Upgrade Tasks

After you complete your upgrade, it is strongly recommended that you use the meta-function [COMPACT_STORAGE](#) to upgrade existing storage bundles to the `.gt` format. Bundling reduces the number of files in your file system by at least 50 percent and improves the performance of file-intensive operations. Improved operations include backups, restores, mergeouts, and moveouts.

Rebuild Geospatial Indexes

Before you upgrade server versions, you must [back up your geospatial indexes](#) in a temporary table. After you upgrade, use [STV_Create_Index](#) with the temporary table as input to rebuild the index, and set the `overwrite` parameter to true to prevent a failure if there is an existing index with the same name:

```
=> SELECT STV_Create_Index(gid, geometry USING PARAMETERS index='pol_idx_new', overwrite=true) OVER()  
FROM temp-table;  
  type | polygons | SRID | min_x | min_y | max_x | max_y | info  
-----+-----+-----+-----+-----+-----+-----+-----  
GEOMETRY | 1       | 0   | 1     | 1     | 3     | 3     |  
(1 row)
```

Now, you can use the index `pol_idx_new` in the Place version.

Upgrading the Streaming Data Scheduler Utility

If you have integrated Vertica with a streaming data application, such as Apache Kafka, you must update the streaming data scheduler utility after you update Vertica.

Note: Schedulers upgraded from version 7.2.x to version 8.0.x are not backwards compatible.

From a command prompt, enter the following command:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --upgrade --upgrade-to-schema schema_name
```

Running the upgrade task more than once has no effect.

For more information on the Scheduler utility, refer to [Scheduler Utility Options](#).

Upgrading Pre-7.2 Backups

In version 7.2 and later, Vertica no longer relies on hard links to perform backups. As a result, pre-7.2 backups are not compatible with later Vertica versions. To resolve this issue, vbr includes the vbr task `7.2_upgrade`. This task copies an existing pre-7.2 backup and creates a 7.2.x-compatible version of it.

Note: Micro Focus recommends that you run this task before performing the first backup of the upgraded database.

To upgrade a backup:

1. Specify the vbr task `7.2_upgrade` in the following form:

```
vbr -t 7.2_upgrade --old-config-file outdated-configfile.ini -c new-configfile.ini
```

2. Verify that the `snapshotName` parameter is the same in the old and new configuration files.

The new configuration file assigns new backup locations for the upgraded backup. This approach preserves the existing backup so you can continue to perform incremental backups on the upgraded backup. After the upgrade is complete, Vertica no longer requires the old configuration file.

If you do not upgrade the backup, the next backup that Vertica executes on the new database is a full backup that subsequently supports incremental backups.

Uninstalling Vertica

For each host in the cluster:

1. Choose a host machine and log in as root (or log in as another user and switch to root).

```
$ su - root  
password: root-password
```

2. Find the name of the package that is installed:

RPM

```
# rpm -qa | grep vertica
```

DEB

```
# dpkg -l | grep vertica
```

3. Remove the package:

RPM

```
# rpm -e package
```

DEB

```
# dpkg -r package
```

Note: If you want to delete the configuration file used with your installation, you can choose to delete the `/opt/vertica/` directory and all subdirectories using this command: `# rm -rf /opt/vertica/`

For each client system:

1. Delete the JDBC driver jar file.
2. Delete ODBC driver data source names.
3. Delete the ODBC driver software:
 - a. In Windows, go to **Start > Control Panel > Add or Remove Programs**.
 - b. Locate Vertica.
 - c. Click **Remove**.

Uninstalling Management Console

The `uninstall` command shuts down Management Console and removes most of the files that MC installation script installed.

To uninstall MC:

1. Log in to the target server as root.
2. Stop Management Console:

```
# /etc/init.d/vertica-console stop
```

For versions of Red Hat 7/CentOS 7 and above, use:

```
# systemctl stop vertica-console
```

3. Look for previously-installed versions of MC and note the version:

RPM

```
# rpm -qa | grep vertica
```

DEB

```
# dpkg -l | grep vertica
```

4. Remove the package:

RPM

```
# rpm -e vertica-console
```

DEB

```
# dpkg -r vertica-console
```

5. Optionally, delete the MC directory and all subdirectories:

```
# rm -rf /opt/vconsole
```

To Reinstall MC

See [Installing and Configuring Management Console](#)

Troubleshooting the Vertica Install

The topics described in this section are performed automatically by the `install_vertica` script and are described in [Installing Vertica](#). If you did not encounter any installation problems, proceed to the [Administrator's Guide](#) for instructions on how to configure and operate a database.

Validation Scripts

Vertica provides several validation utilities that can be used prior to deploying Vertica to help determine if your hosts and network can properly handle the processing and network traffic required by Vertica. These utilities can also be used if you are encountering performance issues and need to troubleshoot the issue.

After you install the Vertica RPM, you have access to the following scripts in `/opt/vertica/bin`:

- [Vcpuperf](#) - a CPU performance test used to verify your CPU performance.
- [Vioperf](#) - an Input/Output test used to verify the speed and consistency of your hard drives.
- [Vnetperf](#) - a Network test used to test the latency and throughput of your network between hosts.

These utilities can be run at any time, but are well suited to use before running the `install_vertica` script.

Vcpuperf

The `vcpuperf` utility measures your server's CPU processing speed and compares it against benchmarks for common server CPUs. The utility performs a CPU test and measures the time it takes to complete the test. The lower the number scored on the test, the better the performance of the CPU.

The `vcpuperf` utility also checks the high and low load times to determine if CPU throttling is enabled. If a server's low-load computation time is significantly longer than the high-load computation time, CPU throttling may be enabled. CPU throttling is a power-saving feature. However, CPU throttling can reduce the performance of your server. Vertica recommends disabling CPU throttling to enhance server performance.

Syntax

```
vcpuperf [-q]
```

Option

Option	Description
-q	Run in quiet mode. Quiet mode displays only the CPU Time, Real Time, and high and low load times.

Returns

- CPU Time: the amount of time it took the CPU to run the test.
- Real Time: the total time for the test to execute.
- High load time: The amount of time to run the load test while simulating a high CPU load.
- Low load time: The amount of time to run the load test while simulating a low CPU load.

Example

The following example shows a CPU that is running slightly slower than the expected time on a Xeon 5670 CPU that has CPU throttling enabled.

```
[root@node1 bin]# /opt/vertica/bin/vcpuperf
Compiled with: 4.1.2 20080704 (Red Hat 4.1.2-52) Expected time on Core 2, 2.53GHz: ~9.5s
Expected time on Nehalem, 2.67GHz: ~9.0s
Expected time on Xeon 5670, 2.93GHz: ~8.0s

This machine's time:
CPU Time: 8.540000s
Real Time:8.710000s

Some machines automatically throttle the CPU to save power.
This test can be done in <100 microseconds (60-70 on Xeon 5670, 2.93GHz).
Low load times much larger than 100-200us or much larger than the corresponding high load time
indicate low-load throttling, which can adversely affect small query / concurrent performance.

This machine's high load time: 67 microseconds.
This machine's low load time: 208 microseconds.
```

Vioperf

The `vioperf` utility quickly tests the performance of your host's input and output subsystem. The utility performs the following tests:

- sequential write
- sequential rewrite
- sequential read
- skip read (read non-contiguous data blocks)

The utility verifies that the host reads the same bytes that it wrote and prints its output to `STDOUT`. The utility also logs the output to a JSON formatted file.

For data in HDFS, the utility tests reads but not writes.

Syntax

```
vioperf [--help] [--duration=<INTERVAL>] [--log-interval=<INTERVAL>]
  [--log-file=<FILE>] [--condense-log] [--thread-count=<N>] [--max-buffer-size=<SIZE>]
  [--preserve-files] [--disable-crc] [--disable-direct-io] [--debug]
  [<DIR>*]
```

Minimum and Recommended I/O Performance

- The minimum required I/O is 20 MB/s read/write per physical processor core on each node, in full duplex (reading and writing) simultaneously, concurrently on all nodes of the cluster.
- The recommended I/O is 40 MB/s per physical core on each node.
- The minimum required I/O rate for a node with 2 hyper-threaded six-core CPUs (12 physical cores) is 240 MB/s. Vertica recommends 480 MB/s.

For example, the I/O rate for a node with 2 hyper-threaded six-core CPUs (12 physical cores) is 240 MB/s required minimum, 480 MB/s recommended.

Options

Option	Description
<code>--help</code>	Prints a help message and exits.
<code>--duration</code>	The length of time <code>vioprobe</code> runs performance tests. The default is 5 minutes. Specify the interval in seconds, minutes, or hours with any of these suffixes: <ul style="list-style-type: none">• Seconds: <code>s</code>, <code>sec</code>, <code>secs</code>, <code>second</code>, <code>seconds</code>. Example: <code>--duration=60sec</code>• Minutes: <code>m</code>, <code>min</code>, <code>mins</code>, <code>minute</code>, <code>minutes</code>. Example: <code>--duration=10min</code>• Hours: <code>h</code>, <code>hr</code>, <code>hrs</code>, <code>hour</code>, <code>hours</code>. Example: <code>--duration=1hrs</code>
<code>--log-interval</code>	The interval at which the log file reports summary information. The default interval is 10 seconds. This option uses the same interval notation as <code>--duration</code> .
<code>--log-file</code>	The path and name where log file contents are written, in JSON. If not specified, then <code>vioperf</code> creates a file named <code>resultsdate-time</code> . JSON in the current directory.
<code>--condense-log</code>	Directs <code>vioperf</code> to write the log file contents in condensed format, one JSON entry per line, rather than as indented JSON syntax.
<code>--thread-count=<N></code>	The number of execution threads to use. By default, <code>vioperf</code> uses all threads available on the host machine.
<code>--max-buffer-size=<SIZE></code>	The maximum size of the in-memory buffer to use for reads or writes. Specify the units with any of these suffixes: <ul style="list-style-type: none">• Bytes: <code>b</code>, <code>byte</code>, <code>bytes</code>.• Kilobytes: <code>k</code>, <code>kb</code>, <code>kilobyte</code>, <code>kilobytes</code>.• Megabytes: <code>m</code>, <code>mb</code>, <code>megabyte</code>, <code>megabytes</code>.• Gigabytes: <code>g</code>, <code>gb</code>, <code>gigabyte</code>, <code>gigabytes</code>.

Option	Description
<code>--preserve-files</code>	Directs <code>vioperf</code> to keep the files it writes. This parameter is ignored for HDFS tests, which are read-only. Inspecting the files can help diagnose write-related failures.
<code>--disable-crc</code>	Directs <code>vioperf</code> to ignore CRC checksums when validating writes. Verifying checksums can add overhead, particularly when running <code>vioperf</code> on slower processors. This parameter is ignored for HDFS tests.
<code>--disable-direct-io</code>	<p>When reading from or writing to a local file system, <code>vioperf</code> goes directly to disk by default, bypassing the operating system's page cache. Using direct I/O allows <code>vioperf</code> to measure performance quickly without having to fill the cache.</p> <p>Disabling this behavior can produce more realistic performance results but slows down the operation of <code>vioperf</code>.</p>
<code>--debug</code>	Directs <code>vioperf</code> to report verbose error messages.
<DIR>	<p>Zero or more directories to test. If you do not specify a directory, <code>vioperf</code> tests the current directory. To test the performance of each disk, specify different directories mounted on different disks.</p> <p>To test reads from a directory on HDFS:</p> <ul style="list-style-type: none">• Use a URL in the <code>hdfs</code> scheme that points to a single directory (not a path) containing files at least 10MB in size. For best results, use 10GB files and verify that there is at least one file per <code>vioperf</code> thread.• If you do not specify a host and port, set the <code>HADOOP_CONF_DIR</code> environment variable to a path including the Hadoop configuration files. This value is the same value that you use for the <code>HadoopConfDir</code> configuration parameter in Vertica. For more information see Configuring the hdfs Scheme.• If the HDFS cluster uses Kerberos, set the <code>HADOOP_USER_NAME</code> environment variable to a Kerberos principal.

Returns

The utility returns the following information:

Heading	Description
test	The test being run (Write, ReWrite, Read, or Skip Read)
directory	The directory in which the test is being run.
counter name	The counter type of the test being run. Can be either MB/s or Seeks per second.
counter value	The value of the counter in MB/s or Seeks per second across all threads. This measurement represents the bandwidth at the exact time of measurement. Contrast with counter value (avg).
counter value (10 sec avg)	The average amount of data in MB/s, or the average number of Seeks per second, for the test being run in the duration specified with --log-interval. The default interval is 10 seconds. The counter value (avg) is the average bandwidth since the last log message, across all threads.
counter value/core	The counter value divided by the number of cores.
counter value/core (10 sec avg)	The counter value (10 sec avg) divided by the number of cores.
thread count	The number of threads used to run the test.
%CPU	The available CPU percentage used during this test.
%IO Wait	The CPU percentage in I/O Wait state during this test. I/O wait state is the time working processes are blocked while waiting for I/O operations to complete.
elapsed time	The amount of time taken for a particular test. If you run the test multiple times, elapsed time increases the next time the test is run.
remaining time	The time remaining until the next test. Based on the --duration option, each of the tests is run at least once. If the test set is run multiple times, then remaining time is how much longer the test will run. The remaining time value is cumulative. Its total is added to elapsed time each time the same test is run again.

Example

Invoking `vioperf` from a terminal outputs the following message and sample results:

```
[dbadmin@v_vmart_node0001 ~]$ /opt/vertica/bin/vioperf --duration=60s
The minimum required I/O is 20 MB/s read and write per physical processor core on each node, in full duplex
i.e. reading and writing at this rate simultaneously, concurrently on all nodes of the cluster.
The recommended I/O is 40 MB/s per physical core on each node.
For example, the I/O rate for a server node with 2 hyper-threaded six-core CPUs is 240 MB/s required minimum, 480 MB/s recommended.

Using direct io (buffer size=1048576, alignment=512) for directory "/home/dbadmin"

test      | directory      | counter name          | counter value | counter value (10 sec avg) |
counter value/core | counter value/core (10 sec avg) | thread count | %CPU | %IO Wait | elapsed
time (s)| remaining time (s)
-----
-----
----
Write     | /home/dbadmin | MB/s                  | 420           | 420           |
210      |              | 210                  | 2             | 89           | 10           |
| 5
Write     | /home/dbadmin | MB/s                  | 412           | 396           |
206      |              | 198                  | 2             | 89           | 9            | 15
| 0
ReWrite   | /home/dbadmin | (MB-read+MB-write)/s | 150+150      | 150+150      |
75+75    |              | 75+75               | 2             | 58           | 40           | 10
| 5
ReWrite   | /home/dbadmin | (MB-read+MB-write)/s | 158+158      | 172+172      |
79+79    |              | 86+86               | 2             | 64           | 33           | 15
| 0
Read      | /home/dbadmin | MB/s                  | 194           | 194           |
97       |              | 97                   | 2             | 69           | 26           | 10
| 5
Read      | /home/dbadmin | MB/s                  | 192           | 190           |
96       |              | 95                   | 2             | 71           | 27           | 15
| 0
SkipRead  | /home/dbadmin | seeks/s              | 659           | 659           |
329.5    |              | 329.5               | 2             | 2            | 85           | 10
| 5
SkipRead  | /home/dbadmin | seeks/s              | 677           | 714           |
338.5    |              | 357                  | 2             | 2            | 59           | 15
| 0
```

Note: When evaluating performance for minimum and recommended I/O, include the Write and Read values in your evaluation. ReWrite and SkipRead values are not relevant to determining minimum and recommended I/O.

Vnetperf

The vnetperf utility allows you to measure the network performance of your hosts. It can measure network latency and the throughput for both the TCP and UDP protocols.

Important: This utility introduces a high network load and must not be used on a running Vertica cluster or database performance is degraded.

Using this utility you can detect:

- if throughput is low for all hosts or a particular host,
- if latency is high for all hosts or a particular host,
- bottlenecks between one or more hosts or subnets,
- too low a limit in the number of TCP connections that can be established simultaneously,
- and if there is a high rate of packet loss on the network.

The latency test measures the latency from the host running the script to the other hosts. Any host that has a particularly high latency should be investigated further.

The throughput tests measure both UDP and TCP throughput. You can specify a rate limit in MB/s to use for these tests, or allow the utility to use a range of throughputs to be used.

Syntax

```
vnetperf [options] [tests]
```

Recommended Network Performance

- The maximum recommended RTT (round-trip time) latency is 1000 microseconds. The ideal RTT latency is 200 microseconds or less. Vertica recommends that clock skew be kept to under 1 second.
- The minimum recommended throughput is 100 MB/s. Ideal throughput is 800 MB/s or more.

Note: UDP numbers may be lower, multiple network switches may reduce performance results.

Options

Option	Description
<code>--condense</code>	Condense the log into one JSON entry per line, instead of indented JSON syntax.
<code>--collect-logs</code>	Collect the test log files from each host.
<code>--datarate <i>rate</i></code>	Limit the throughput to this rate in MB/s. A rate of 0 loops the tests through several different rates. The default is 0.
<code>--duration <i>seconds</i></code>	The time limit for each test to run in seconds. The default is 1.
<code>--hosts <i>host1,host2,...</i></code>	A comma-separated list of hosts on which to run the tests. Do not use spaces between the comma's and the host names.
<code>--hosts <i>file</i></code>	A hosts file that specifies the hosts on which to run the tests. If the <code>--hosts</code> argument is not used, then the utility attempts to access admintools and determine the hosts in the cluster.
<code>--identity-file <i>file</i></code>	If using passwordless SSH/SCP access between the hosts, then specify the key file used to gain access to the hosts.
<code>--ignore-bad-hosts</code>	If set, run the tests on the reachable hosts even if some hosts are not reachable. If not set, and a host is unreachable, then no tests are run on any hosts.
<code>--log-dir <i>directory</i></code>	If <code>--collect-logs</code> is set, the directory in which to place the collected logs. The default directory is named <code>logs.netperf.<timestamp></code>
<code>--log-level <i>LEVEL</i></code>	The log level to use. Possible values are: INFO, ERROR, DEBUG, and WARN. The default is WARN.
<code>--list-tests</code>	Lists the tests that can be run by this utility.
<code>--output-file <i>file</i></code>	The file that JSON results are written to. The default is <code>results.<timestamp>.json</code> .

Option	Description
<code>--ports port1,port2,port3</code>	The port numbers to use. If only one is specified then the next two numbers in sequence are also used. The default ports are 14159,14160, 14161.
<code>--scp-options 'options'</code>	Using this argument, you can specify one or more standard SCP command line arguments enclosed in single quotes. SCP is used to copy test binaries over to the target hosts.
<code>--ssh-options 'options'</code>	Using this argument, you can specify one or more standard SSH command line arguments enclose in single quotes. SSH is used to issue test commands on the target hosts.
<code>--vertica-install directory</code>	If specified, then the utility assumes Vertica is installed on each of the hosts and to use the test binaries on the target system rather than copying them over using SCP.

Tests

Note: If the tests argument is omitted then all tests are run.

Test	Description
latency	Test the latency to each of the hosts.
tcp-throughput	Test the TCP throughput amongst the hosts.
udp-throughput	Test the UDP throughput amongst the hosts.

Returns

For each host it returns the following:

Latency test returns:

- The Round Trip Time (rtt) latency for each host in milliseconds.
- Clock Skew = the difference in time shown by the clock on the target host relative to the host running the utility.

UDP and TCP throughput tests return:

- The date/time and test name.
- The rate limit in MB/s.
- The node being tested.
- Sent and Received data in MB/s and bytes.
- The duration of the test in seconds.

Example

```
/opt/vertica/bin/vnetperf --condense -hosts 10.20.100.66,10.20.100.67 --identity-file  
'/root/.ssh/vid_rsa'
```

Enable Secure Shell (SSH) Logins

The administrative account must be able to use Secure Shell (SSH) to log in (ssh) to all hosts without specifying a password. The shell script `install_vertica` does this automatically. This section describes how to do it manually if necessary.

1. If you do not already have SSH installed on all hosts, log in as root on each host and install it now. You can download a free version of the SSH connectivity tools from [OpenSSH](#).
2. Log in to the Vertica administrator account (dbadmin in this example).
3. Make your home directory (~) writable only by yourself. Choose one of:

```
$ chmod 700 ~
```

or

```
$ chmod 755 ~
```

where:

700 includes	755 includes
400 read by owner	400 read by owner

200 write by owner	200 write by owner
100 execute by owner	100 execute by owner
	040 read by group
	010 execute by group
	004 read by anybody (other)
	001 execute by anybody

4. Change to your home directory:

```
$ cd ~
```

5. Generate a private key/ public key pair:

```
$ ssh-keygen -t rsaGenerating public/private rsa key pair.  
Enter file in which to save the key (/home/dbadmin/.ssh/id_rsa):  
Created directory '/home/dbadmin/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/dbadmin/.ssh/id_rsa.  
Your public key has been saved in /home/dbadmin/.ssh/id_rsa.pub.
```

6. Make your .ssh directory readable and writable only by yourself:

```
$ chmod 700 ~/.ssh
```

7. Change to the .ssh directory:

```
$ cd ~/.ssh
```

8. Copy the file id_rsa.pub onto the file authorized_keys2.

```
$ cp id_rsa.pub authorized_keys2
```

9. Make the files in your .ssh directory readable and writable only by yourself:

```
$ chmod 600 ~/.ssh/*
```

10. For each cluster host:

```
$ scp -r ~/.ssh <host>:.
```

11. Connect to each cluster host. The first time you ssh to a new remote machine, you could get a message similar to the following:

```
$ ssh dev0 Warning: Permanently added 'dev0,192.168.1.92' (RSA) to the list of known hosts.
```

This message appears only the first time you ssh to a particular remote host.

See Also

- [OpenSSH](#)

Upgrading Your Operating System on Nodes in Your Vertica Cluster

If you need to upgrade the operating system on the nodes in your Vertica cluster, check with the documentation for your Linux distribution to make sure they support the particular upgrade you are planning.

For example, the following articles provide information about upgrading Red Hat:

- [How do I upgrade from Red Hat Enterprise Linux 6 to Red Hat Enterprise Linux 7?](#)
- [Does Red Hat support upgrades between major versions of Red Hat Enterprise Linux?](#)

After you confirm that you can perform the upgrade, follow the steps at [Best Practices for Upgrading the Operating System on Nodes in a Vertica Cluster](#).

Upgrading the Operating System on the Nodes in Your Vertica Cluster

For information about upgrading the operating system on the nodes in your Vertica cluster, see [Best Practices for Upgrading the Operating System on Nodes in a Vertica Cluster](#).

Appendix: Time Zones

- [Using Time Zones With Vertica](#)
- [Africa](#)
- [America](#)
- [Antarctica](#)
- [Asia](#)
- [Atlantic](#)
- [Australia](#)
- [Etc/GMT](#)
- [Europe](#)
- [Indian](#)
- [Pacific](#)

Using Time Zones With Vertica

Vertica uses the TZ environment variable on each node, if it has been set, for the default current time zone. Otherwise, Vertica uses the operating system time zone.

The TZ variable can be set by the operating system during login (see `/etc/profile`, `/etc/profile.d`, or `/etc/bashrc`) or by the user in `.profile`, `.bashrc` or `.bash-profile`.

TZ must be set to the same value on each node when you start Vertica.

The following command returns the current time zone for your database:

```
=> SHOW TIMEZONE;
  name |      setting
-----+-----
timezone | America/New_York
(1 row)
```

You can also use the `SET TIMEZONE TO { value | 'value' }` command to set the time zone for a single session.

There is no database default time zone; instead, `TIMESTAMP WITH TIMEZONE (TIMESTAMPTZ)` data is stored in GMT (UTC) by converting data from the current local time zone to GMT.

When `TIMESTAMPTZ` data is used, data is converted back to use the current local time zone, which might be different from the local time zone where the data was stored. This conversion takes into account Daylight Saving Time (Summer Time), if applicable, depending on the year and date, to know when the Daylight Saving Time change occurred.

`TIMESTAMP WITHOUT TIMEZONE` data stores the timestamp, as given, and retrieves it exactly as given. The current time zone is ignored. The same is true for `TIME WITHOUT TIMEZONE`. For `TIME WITH TIMEZONE (TIMETZ)`, however, the current time zone setting is stored along with the given time, and that time zone is used on retrieval.

Note: Micro Focus recommends that you use `TIMESTAMPTZ`, not `TIMETZ`.

`TIMESTAMPTZ` uses the current time zone on both input and output, such as in the following example:

```
=> CREATE TEMP TABLE s (tstz TIMESTAMPTZ);=> SET TIMEZONE TO 'America/New_York';
=> INSERT INTO s VALUES ('2009-02-01 00:00:00');
=> INSERT INTO s VALUES ('2009-05-12 12:00:00');
=> SELECT tstz AS 'Local timezone', tstz AT TIMEZONE 'America/New_York' AS 'America/New_York',
      tstz AT TIMEZONE 'GMT' AS 'GMT' FROM s;
```

Local timezone	America/New_York	GMT
2009-02-01 00:00:00-05	2009-02-01 00:00:00	2009-02-01 05:00:00
2009-05-12 12:00:00-04	2009-05-12 12:00:00	2009-05-12 16:00:00

(2 rows)

The `-05` in the Local time zone column above shows that the data is displayed in EST, while `-04` indicates EDT. The other two columns show the `TIMESTAMP WITHOUT TIMEZONE` at the specified time zone.

The next example illustrates what occurs if the current time zone is changed to, for example, Greenwich Mean Time:

```
=> SET TIMEZONE TO 'GMT';=> SELECT tstz AS 'Local timezone', tstz AT TIMEZONE 'America/New_York' AS
      'America/New_York', tstz AT TIMEZONE 'GMT' as 'GMT' FROM s;
```

Local timezone	America/New_York	GMT
2009-02-01 05:00:00+00	2009-02-01 00:00:00	2009-02-01 05:00:00
2009-05-12 16:00:00+00	2009-05-12 12:00:00	2009-05-12 16:00:00

(2 rows)

The `+00` in the Local time zone column above indicates that `TIMESTAMPTZ` is displayed in 'GMT'.

The approach of using TIMESTAMPTZ fields to record events captures the GMT of the event, as expressed in terms of the local time zone. Later, it allows for easy conversion to any other time zone, either by setting the local time zone or by specifying an explicit AT TIMEZONE clause.

The following example shows how TIMESTAMP WITHOUT TIMEZONE fields work in Vertica.

```
=> CREATE TEMP TABLE tnoz (ts TIMESTAMP);=> INSERT INTO tnoz VALUES('2009-02-01 00:00:00');
=> INSERT INTO tnoz VALUES('2009-05-12 12:00:00');
=> SET TIMEZONE TO 'GMT';
=> SELECT ts AS 'No timezone', ts AT TIMEZONE 'America/New_York' AS
       'America/New_York', ts AT TIMEZONE 'GMT' AS 'GMT' FROM tnoz;
       No timezone      | America/New_York      | GMT
-----+-----+-----
2009-02-01 00:00:00 | 2009-02-01 05:00:00+00 | 2009-02-01 00:00:00+00
2009-05-12 12:00:00 | 2009-05-12 16:00:00+00 | 2009-05-12 12:00:00+00
(2 rows)
```

The +00 at the end of a timestamp indicates that the setting is TIMESTAMP WITH TIMEZONE in GMT (the current time zone). The 'America/New_York' column shows what the 'GMT' setting was when you recorded the time, assuming you read a normal clock in the time zone 'America/New_York'. What this shows is that if it is midnight in the 'America/New_York' time zone, then it is 5 am GMT.

Note: 00:00:00 Sunday February 1, 2009 in America/New_York converts to 05:00:00 Sunday February 1, 2009 in GMT.

The 'GMT' column displays the GMT time, assuming the input data was captured in GMT.

If you don't set the time zone to GMT, and you use another time zone, for example 'America/New_York', then the results display in 'America/New_York' with a -05 and -04, showing the difference between that time zone and GMT.

```
=> SET TIMEZONE TO 'America/New_York';
=> SHOW TIMEZONE;
       name | setting
-----+-----
timezone | America/New_York
(1 row)
=> SELECT ts AS 'No timezone', ts AT TIMEZONE 'America/New_York' AS
       'America/New_York', ts AT TIMEZONE 'GMT' AS 'GMT' FROM tnoz;
       No timezone      | America/New_York      | GMT
-----+-----+-----
2009-02-01 00:00:00 | 2009-02-01 00:00:00-05 | 2009-01-31 19:00:00-05
2009-05-12 12:00:00 | 2009-05-12 12:00:00-04 | 2009-05-12 08:00:00-04
(2 rows)
```

In this case, the last column is interesting in that it returns the time in New York, given that the data was captured in 'GMT'.

See Also

- [TZ Environment Variable](#)
- [SET TIME ZONE](#)
- [Date/Time Data Types](#)

Africa

Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo Egypt
Africa/Casablanca
Africa/Ceuta
Africa/Conakry

Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu

Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli Libya
Africa/Tunis
Africa/Windhoek

America

America/Adak America/Atka US/Aleutian
America/Anchorage SystemV/YST9YDT US/Alaska
America/Anguilla
America/Antigua
America/Araguaina
America/Aruba
America/Asuncion
America/Bahia
America/Barbados

America/Belem
America/Belize
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago CST6CDT SystemV/CST6CDT US/Central
America/Chihuahua
America/Cordoba America/Rosario
America/Costa_Rica
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson
America/Dawson_Creek
America/Denver MST7MDT SystemV/MST7MDT US/Mountain America/Shiprock Navajo
America/Detroit US/Michigan

America/Dominica
America/Edmonton Canada/Mountain
America/Eirunepe
America/El_Salvador
America/Ensenada America/Tijuana Mexico/BajaNorte
America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax Canada/Atlantic SystemV/AST4ADT
America/Havana Cuba
America/Hermosillo
America/Indiana/Indianapolis
America/Indianapolis
America/Fort_Wayne EST SystemV/EST5 US/East-Indiana
America/Indiana/Knox America/Knox_IN US/Indiana-Starke
America/Indiana/Marengo
America/Indiana/Vevay
America/Inuvik

America/Iqaluit
America/Jamaica Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville America/Louisville
America/Kentucky/Monticello
America/La_Paz
America/Lima
America/Los_Angeles PST8PDT SystemV/PST8PDT US/Pacific US/Pacific- New
America/Maceio
America/Managua
America/Manaus Brazil/West
America/Martinique
America/Mazatlan Mexico/BajaSur
America/Mendoza
America/Menominee
America/Merida
America/Mexico_City Mexico/General
America/Miquelon
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau

America/New_York EST5EDT SystemV/EST5EDT US/Eastern
America/Nipigon
America/Nome
America/Noronha Brazil/DeNoronha
America/North_Dakota/Center
America/Panama
America/Pangnirtung
America/Paramaribo
America/Phoenix MST SystemV/MST7 US/Arizona
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre America/Rio_Branco Brazil/Acre
America/Porto_Velho
America/Puerto_Rico SystemV/AST4
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina Canada/East-Saskatchewan Canada/Saskatchewan SystemV/CST6
America/Santiago Chile/Continental
America/Santo_Domingo
America/Sao_Paulo Brazil/East
America/Scoresbysund
America/St_Johns Canada/Newfoundland
America/St_Kitts

America/St_Lucia
America/St_Thomas America/Virgin
America/St_Vincent
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Toronto Canada/Eastern
America/Tortola
America/Vancouver Canada/Pacific
America/Whitehorse Canada/Yukon
America/Winnipeg Canada/Central
America/Yakutat
America/Yellowknife

Antarctica

Antarctica/Casey
Antarctica/Davis
Antarctica/DumontDUrville
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/South_Pole
Antarctica/Palmer

Antarctica/Rothera
Antarctica/Syowa
Antarctica/Vostok

Asia

Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat Asia/Ashkhabad
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Choibalsan
Asia/Chongqing Asia/Chungking
Asia/Colombo

Asia/Dacca Asia/Dhaka
Asia/Damascus
Asia/Dili
Asia/Dubai
Asia/Dushanbe
Asia/Gaza
Asia/Harbin
Asia/Hong_Kong Hongkong
Asia/Hovd
Asia/Irkutsk
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem Asia/Tel_Aviv Israel
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Kashgar
Asia/Katmandu
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuching
Asia/Kuwait
Asia/Macao Asia/Macau
Asia/Magadan

Asia/Makassar Asia/Ujung_Pandang
Asia/Manila
Asia/Muscat
Asia/Nicosia Europe/Nicosia
Asia/Novosibirsk
Asia/Omsk
Asia/Oral
Asia/Phnom_Penh
Asia/Pontianak
Asia/Pyongyang
Asia/Qatar
Asia/Qyzylorda
Asia/Rangoon
Asia/Riyadh
Asia/Riyadh87 Mideast/Riyadh87
Asia/Riyadh88 Mideast/Riyadh88
Asia/Riyadh89 Mideast/Riyadh89
Asia/Saigon
Asia/Sakhalin
Asia/Samarkand
Asia/Seoul ROK
Asia/Shanghai PRC
Asia/Singapore Singapore
Asia/Taipei ROC

Asia/Tashkent
Asia/Tbilisi
Asia/Tehran Iran
Asia/Thimbu Asia/Thimphu
Asia/Tokyo Japan
Asia/Ulaanbaatar Asia/Ulan_Bator
Asia/Urumqi
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yekaterinburg
Asia/Yerevan

Atlantic

Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Madeira
Atlantic/Reykjavik Iceland
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley

Australia

Australia/ACT
Australia/Canberra
Australia/NSW
Australia/Sydney
Australia/Adelaide
Australia/South
Australia/Brisbane
Australia/Queensland
Australia/Broken_Hill
Australia/Yancowinna
Australia/Darwin
Australia/North
Australia/Hobart
Australia/Tasmania
Australia/LHI
Australia/Lord_Howe
Australia/Lindeman
Australia/Melbourne
Australia/Victoria
Australia/Perth Australia/West

Etc/GMT

Etc/GMT+0...Etc/GMT+12

Etc/GMT-0...ETC/GMT-14

Europe

Europe/Amsterdam

Europe/Andorra

Europe/Athens

Europe/Belfast

Europe/Belgrade

Europe/Ljubljana

Europe/Sarajevo

Europe/Skopje

Europe/Zagreb

Europe/Berlin

Europe/Brussels

Europe/Bucharest

Europe/Budapest

Europe/Chisinau Europe/Tiraspol

Europe/Copenhagen

Europe/Dublin Eire

Europe/Gibraltar

Europe/Helsinki
Europe/Istanbul Asia/Istanbul Turkey
Europe/Kaliningrad
Europe/Kiev
Europe/Lisbon Portugal
Europe/London GB GB-Eire
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Minsk
Europe/Monaco
Europe/Moscow W-SU
Europe/Oslo Arctic/Longyearbyen Atlantic/Jan_Mayen
Europe/Paris
Europe/Prague Europe/Bratislava
Europe/Riga
Europe/Rome Europe/San_Marino Europe/Vatican
Europe/Samara
Europe/Simferopol
Europe/Sofia
Europe/Stockholm
Europe/Tallinn
Europe/Tirane

Europe/Uzhgorod
Europe/Vaduz
Europe/Vienna
Europe/Vilnius
Europe/Warsaw Poland
Europe/Zaporozhye
Europe/Zurich

Indian

Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives
Indian/Mauritius
Indian/Mayotte
Indian/Reunion

Pacific

Pacific/Apia
Pacific/Auckland NZ
Pacific/Chatham NZ-CHAT
Pacific/Easter Chile/EasterIsland
Pacific/Efate
Pacific/Enderbury
Pacific/Fakaofu
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier SystemV/YST9
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu HST SystemV/HST10 US/Hawaii
Pacific/Johnston
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Kwajalein Kwajalein
Pacific/Majuro
Pacific/Marquesas

Pacific/Midway
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Samoa US/Samoa
Pacific/Palau
Pacific/Pitcairn SystemV/PST8
Pacific/Ponape
Pacific/Port_Moresby
Pacific/Rarotonga
Pacific/Saipan
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
Pacific/Yap

Getting Started

Welcome to Getting Started. This guide serves as a tutorial, walking you through the process of configuring an Vertica Analytics Platform database and running example queries.

Prerequisites

Before you start, Micro Focus recommends that you read [Vertica Concepts](#) to gain a quick understanding of unfamiliar concepts.

Using This Guide

Getting Started shows how to set up a Vertica database and run simple queries that perform common database tasks.

Who Should Use This Guide?

Getting Started targets anyone who wants to learn how to create and run a Vertica database. This guide requires no special knowledge at this point, although a rudimentary knowledge of basic SQL commands is useful when you begin to run queries.

What You Need

The examples in this guide require one of the following:

- Vertica installed on one host or a cluster of hosts. Vertica recommends a minimum of three hosts in the cluster.
- Vertica installed on a virtual machine (VM).

For further instructions about installation, see [Installing Vertica](#).

Accessing Your Database

You access your database with an SSH client or the terminal utility in your Linux console, such as `vsql`. Throughout this guide, you use the following user interfaces:

- Linux command line (shell) interface
- [Vertica Administration Tools](#)
- [vsql client interface](#)
- [Vertica Management Console](#)

Downloading and Starting the Vertica Community Edition VM

For a hands-on introduction to Vertica, you can use the Vertica Community Edition Virtual Machine (Vertica CE VM), which is available for download on my.vertica.com.

The Vertica CE VM is a pre-configured Linux environment that includes the Vertica Analytic Database and client tools. After you download and startup the VM, you no longer need internet access. Everything you need is available locally within the VM environment. The VM includes a user guide that steps you through a set of typical tasks involved in administering your Vertica database and querying data.

Requirements

The Vertica CE VM is packaged as an Open Virtualization Architecture (OVA) file that can be imported into a virtualization application that supports the OVA format. If you do not already have a virtualization product, you can download and install one of the free VM players, such as:

- VMWare Workstation Player
- Oracle VirtualBox

The Vertica CE VM requires between 20 and 50 GB of available disk space on the host computer. The OVA file alone is about 8 GB on download. When you import the OVA file into the VM player, it is uncompressed to about 16 GB. The VM can use up to a maximum of 50 GB, depending on the operations you perform.

Download the Vertica CE VM

1. Sign in or create an account on my.vertica.com. You must be logged in to download the VM.
2. On the Downloads page, locate the Vertica CE VM OVA file, the *Vertica CE VM User Guide* and the short introductory video.

- a. Watch the video for a quick overview.
- b. Download the user guide if you would like to save it outside the VM. Otherwise, you can use the copy of the user guide that is included in the VM itself.
- c. Download the OVA file.

Note: Because of its size, the OVA file takes some time to download.

Start Up the Vertica CE VM

1. Start your VM player.
2. In the VM player, select **Open** (VMware) or **Import** (VirtualBox) and select the Vertica CE VM OVA file. This initial load process may take some time.
3. Start the Vertica CE VM.
4. Log in to the Vertica CE VM:
User name: **Vertica DBA**

Password: **password**

What's In the VM?

In the Vertica CE VM, you will find these components up and running and ready for use:

- Vertica Analytic Database Community Edition
- Vertica VMart example database
- Vertica Management Console
- Vertica Administration Tools
- vsql
- The complete Vertica documentation set
- The *Vertica CE VM User Guide*

Note: VMart is fully loaded, except for the `online_sales` fact table and its two unique dimensions. Instructions for loading the missing data are provided as an exercise in the *Vertica CE VM User Guide*.

The Vertica CE VM Environment

The Vertica CE VM is a CentOS 7.3 Virtual Machine. The VM is configured with the following settings:

- 2 CPUs
- 4 GB of RAM
- 50 GB HDD

The Vertica Community Edition database that is running in the VM has a single node.

Note: The Vertica VM is not supported for production use.

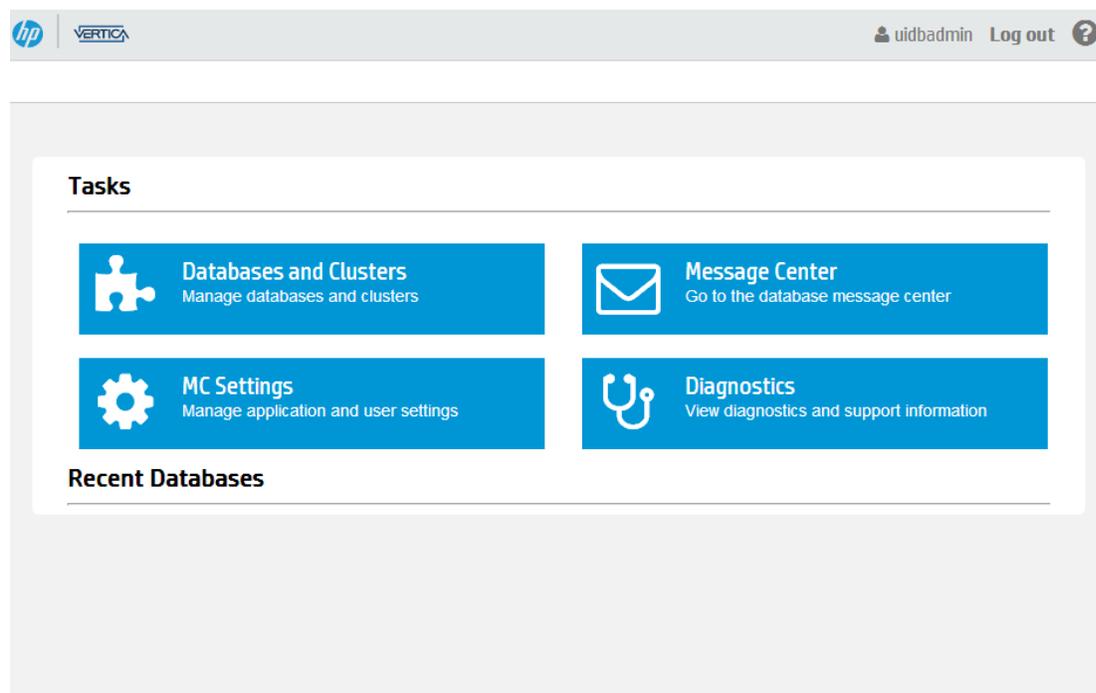
Using the Vertica Interfaces

Vertica provides a set of tools that allows you to perform administrative tasks quickly and easily. The administration tasks in Vertica can be done using the Management Console (MC) or the Administration Tools. The MC provides a unified view of your Vertica cluster through a browser connection, while the Administration Tools are implemented using Dialog, a graphical user interface that works in terminal (character-cell) windows.

Using Management Console

The Management Console provides some, but not all, of the functionality that the Administration Tools provides. In addition, the MC provides extended functionality not available in the Administration Tools, such as a graphical view of your Vertica database and detailed monitoring charts and graphs.

Most of the information you need to use MC is available on the MC interface, as seen in the following two screenshots. For installation instructions, see [Installing and Configuring Management Console](#) in the Installation Guide. For an introduction to MC functionality, architecture, and security, see [Management Console](#) in [Vertica Concepts](#).



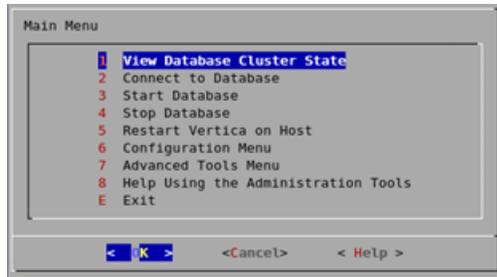
Configuration	Unix user (for application server)	uidbadadmin
SSL certificates	Unix user group (for application server)	uidbadadmin
Authentication	HP Vertica user home path	/home/uidbadadmin
User management		
Resource access	HP Vertica license path	<input type="text" value="/home/uidbadadmin"/>
HP Vertica Installation	HP Vertica database catalog path	<input type="text" value="/home/uidbadadmin"/>
Theme	HP Vertica database data path	<input type="text" value="/home/uidbadadmin"/>
	HP Vertica database temp path	<input type="text" value="/home/uidbadadmin"/>
	Application server running port	<input type="text" value="5450"/>
	Default HP Vertica agent port	<input type="text" value="5444"/>

Running the Administration Tools

A man page is available for convenient access to Administration Tools details. If you are running as the dbadmin user, type `man admintools`. If you are running as a different user, type `man -M /opt/vertica/man admintools`. If possible, always run the Administration Tools using the database administrator account (dbadmin) on the administration host.

The Administration Tools interface responds to mouse clicks in some terminal windows, particularly local Linux windows, but you might find that it responds only to keystrokes. For a quick reference to keystrokes, see [Using Keystrokes in the Administration Tools Interface](#) in this guide.

When you run Administration Tools, the **Main Menu** dialog box appears with a dark blue background and a title on top. The screen captures used in this documentation set are cropped down to the dialog box itself, as shown in the following screenshot.



First Time Only

The first time you log in as the database administrator and run the Administration Tools, complete the following steps.

1. In the EULA (end-user license agreement) window, type `accept` to proceed. A window displays, requesting the location of the license key file you downloaded from the Micro Focus Web site. The default path is `/tmp/vlicense.dat`.
2. Type the absolute path to your license key (for example, `/tmp/vlicense.dat`) and click **OK**.
3. To return to the command line, select **Exit** and click **OK**.

Using Keystrokes in the Administration Tools Interface

The following table is a quick reference to keystroke usage in the Administration Tools interface. See [Using the Administration Tools](#) in the Administrator's Guide for full details.

Return	Run selected command.
Tab	Cycle between OK , Cancel , Help , and menu.
Up/Down Arrow	Move cursor up and down in menu, window, or help file.
Space	Select item in list.
Character	Select corresponding command from menu.

Introducing the VMart Example Database

Vertica ships with a sample multi-schema database called the VMart Example Database, which represents a database that might be used by a large supermarket (VMart) to access information about its products, customers, employees, and online and physical stores. Using this example, you can create, run, optimize, and test a multi-schema database.

The VMart database contains the following schema:

- `public` (automatically created in any newly created Vertica database)
- `store`
- `online_Sales`

VMart Database Location and Scripts

If you installed Vertica from the RPM package, the VMart schema is installed in the `/opt/vertica/examples/VMart_Schema` directory. This folder contains the following script files that you can use to get started quickly. Use the scripts as templates for your own applications.

Script/file name	Description
<code>vmart_count_data.sql</code>	SQL script that counts rows of all example database tables, which you can use to verify load.
<code>vmart_define_schema.sql</code>	SQL script that defines the logical schema for each table and referential integrity constraints.
<code>vmart_gen.cpp</code>	Data generator source code (C++).
<code>vmart_gen</code>	Data generator executable

	file.
<code>vmart_load_data.sql</code>	SQL script that loads the generated sample data to the corresponding tables using COPY DIRECT.
<code>vmart_queries.sql</code>	SQL script that contains concatenated sample queries for use as a training set for the Database Designer.
<code>vmart_query_##.sql</code>	SQL scripts that contain individual queries; for example, <code>vmart_query_01</code> through <code>vmart_query_09.sql</code>
<code>vmart_schema_drop.sql</code>	SQL script that drops all example database tables.

For more information about the schema, tables, and queries included with the VMart example database, see the [Appendix](#).

Installing and Connecting to the VMart Example Database

Follow the steps in this section to create the fully functioning, multi-schema VMart example database that you'll use to run sample queries. The number of example databases you create within a single Vertica installation is limited only by the disk space available on your system; however, Micro Focus strongly recommends that you start only one example database at a time to avoid unpredictable results.

Vertica provides two options to install the example database:

- A quick installation that lets you create the example database and start using it immediately. See [Quick Installation Using a Script](#) in this guide for details. Use this method to bypass the schema and table creation processes and start querying immediately.
- An advanced-but-simple example database installation using the Administration Tools interface. See [Advanced Installation](#) in this guide for details. Use this method to better understand the database creation process and practice creating schema and tables, and loading data.

Note: Both installation methods create a database named VMart. If you try both installation methods, you will either need to drop the VMart database you created (see [Restoring the Status of Your Host](#) in this guide) or create the subsequent database with a new name. However, Micro Focus strongly recommends that you start only one example database at a time to avoid unpredictable results

This tutorial uses Vertica-provided queries, but you can follow the same set of procedures later, when you create your own design and use your own queries file.

After you install the VMart database, the database has started. Connect to it using the steps in [Step 3: Connecting to the Database](#).

Quick Installation Using a Script

The script you need to perform a quick installation is located in `/opt/vertica/sbin` and is called `install_example`. This script creates a database on the default port (5433), generates data, creates the schema and a default superprojection, and loads the data. The folder also contains a `delete_example` script, which stops and drops the database.

1. In a terminal window, log in as the database administrator.

```
$ su dbadmin  
Password: (your password)
```

2. Change to the /examples directory.

```
$ cd /opt/vertica/examples
```

3. Run the install script:

```
$ /opt/vertica/sbin/install_example VMart
```

After installation, you should see the following:

```
[dbadmin@localhost examples]$ /opt/vertica/sbin/install_example VMart  
Installing VMart example example database  
Mon Jul 22 06:57:40 PDT 2013  
Creating Database  
Completed  
Generating Data. This may take a few minutes.  
Completed  
Creating schema  
Completed  
Loading 5 million rows of data. Please stand by.  
Completed  
Removing generated data files  
Example data
```

The example database log files, `ExampleInstall.txt` and `ExampleDelete.txt`, are written to `/opt/vertica/examples/log`.

To start using your database, continue to [Connecting to the Database](#) in this guide. To drop the example database, see [Restoring the Status of Your Host](#) in this guide.

Advanced Installation

To perform an advanced-but-simple installation, set up the VMart example database environment and then create the database using the Administration Tools or Management Console.

Note: If you installed the VMart database using the quick installation method, you cannot complete the following steps because the database has already been created.

To try the advanced installation, drop the example database (see [Restoring the Status of Your Host](#) on this guide) and perform the advanced Installation, or create a new example

database with a different name. However, Micro Focus strongly recommends that you install only one example database at a time to avoid unpredictable results.

The advanced installation requires the following steps:

- [Step 1: Setting Up the Example Environment](#)
- [Step 2: Creating the Example Database](#)
- [Step 3: Connecting to the Database](#)
- [Step 4: Defining the Database Schema](#)
- [Step 5: Loading Data](#)

Step 1: Setting Up the Example Environment

1. Stop all databases running on the same host on which you plan to install your example database.

If you are unsure if other databases are running, run the Administration Tools and select **View Cluster State**. The State column should show DOWN values on pre-existing databases.

If databases are running, click **Stop Database** in the **Main Menu** of the Administration Tools interface and click **OK**.

2. In a terminal window, log in as the database administrator:

```
$ su dbadmin  
Password:
```

3. Change to the `/VMart_Schema` directory.

```
$ cd /opt/vertica/examples/VMart_Schema
```

Do not change directories while following this tutorial. Some steps depend on being in a specific directory.

4. Run the sample data generator.

```
$ ./vmart_gen
```

5. Let the program run with the default parameters, which you can review in the README file.

```
Using default parameters
datadirectory = ./
numfiles = 1
seed = 2
null = ' '
timefile = Time.txt
numfactsalesrows = 5000000
numfactorderrows = 300000
numprodkeys = 60000
numstorekeys = 250
numpromokeys = 1000
numvendkeys = 50
numcustkeys = 50000
numempkeys = 10000
numwarehousekeys = 100
numshippingkeys = 100
numonlinepagekeys = 1000
numcallcenterkeys = 200
numfactonlinesalesrows = 5000000
numinventoryfactrows = 300000
gen_load_script = false
Data Generated successfully !

Using default parameters
datadirectory = ./
numfiles = 1
seed = 2
null = ' '
timefile = Time.txt
numfactsalesrows = 5000000
numfactorderrows = 300000
numprodkeys = 60000
numstorekeys = 250
numpromokeys = 1000
numvendkeys = 50
numcustkeys = 50000
numempkeys = 10000
numwarehousekeys = 100
numshippingkeys = 100
numonlinepagekeys = 1000
numcallcenterkeys = 200
numfactonlinesalesrows = 5000000
numinventoryfactrows = 300000
gen_load_script = false
Data Generated successfully !
```

6. If the `vmart_gen` executable does not work correctly, recompile it as follows, and run the sample data generator script again.

```
$ g++ vmart_gen.cpp -o vmart_gen  
$ chmod +x vmart_gen  
$ ./vmart_gen
```

Step 2: Creating the Example Database

To create the example database: use the Administration Tools or Management Console, as described in this section.

Creating the Example Database Using the Administration Tools

In this procedure, you create the example database using the Administration Tools. To use the Management Console, go to the next section.

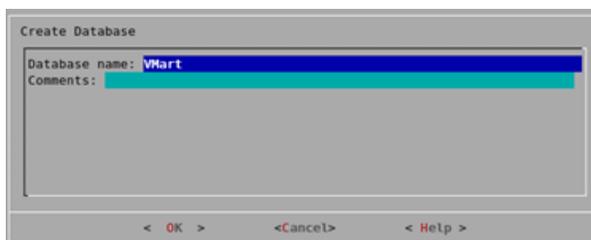
Note: If you have not used Administration Tools before, see [Running the Administration Tools](#) in this guide.

1. Run the Administration Tools.

```
$ /opt/vertica/bin/admintools
```

or simply type `admintools`

2. From the Administration Tools **Main Menu**, click **Configuration Menu** and click **OK**.
3. Click **Create Database** and click **OK**.
4. Name the database **VMart** and click **OK**.

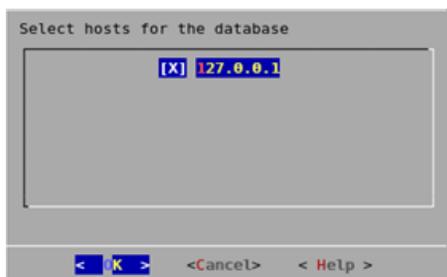


5. Click **OK** to bypass the password and click **Yes** to confirm.

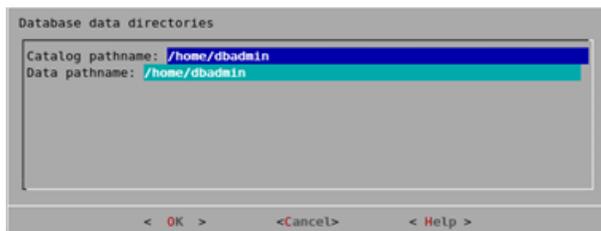
There is no need for a database administrator password in this tutorial. When you create a production database, however, always specify an administrator password. Otherwise, the database is permanently set to trust authentication (no passwords).

6. Select the hosts you want to include from your Vertica cluster and click **OK**.

This example creates the database on a one-host cluster. Micro Focus recommends a minimum of three hosts in the cluster. If you are using the Vertica Community Edition, you are limited to three nodes.



7. Click **OK** to select the default paths for the data and catalog directories.

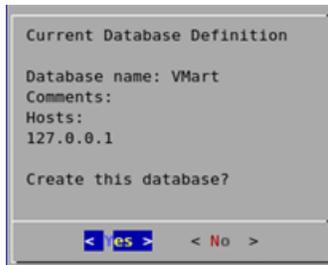


- Catalog and data paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.
- When you create a production database, you'll likely specify other locations than the default. See [Prepare Disk Storage Locations](#) in the Administrator's Guide for more information.

8. Since this tutorial uses a one-host cluster, a K-safety warning appears. Click **OK**.

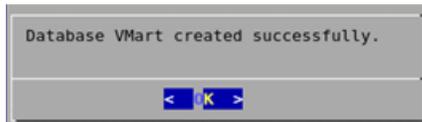


9. Click **Yes** to create the database.



During database creation, Vertica automatically creates a set of node definitions based on the database name and the names of the hosts you selected and returns a success message.

10. Click **OK** to close the **Database VMart created successfully** message.



Creating the Example Database Using Management Console

In this procedure, you create the example database using Management Console. To use the Administration Tools, follow the steps in the preceding section.

Note: To use Management Console, the console should already be installed and you should be familiar with its concepts and layout. See [Using Management Console](#) in this guide for a brief overview, or for detailed information, see [Management Console](#) in [Vertica Concepts](#) and [Installing and Configuring Management Console](#) in [Installing Vertica](#).

1. Connect to Management Console and log in.
2. On the Home page, under **Manage Information**, click **Existing Infrastructure** to go to the Databases and Clusters page.

Prepare data and databases

Provision Databases
Create, import and connect

Manage Information

Existing Infrastructure
Manage databases and clusters

Message Center
Go to the database message center

Diagnostics
View diagnostics and support information

MC Settings
Manage application and user settings

Recent Databases


VMart
(10.20.100.247)

3. Click to select the appropriate existing cluster and click **Create Database**.

Home > Databases and Clusters

Infrastructure

hp | VERTICA

Vertica Clusters: 14
Vertica Databases: 14

Clusters

143567..
Type: Cluster
Size (# nodes): 3

Databases

VMart (1...
Type: Database
Status: Down

VMart null
Type: Database
Status: Up

Cluster Details Dialog:

Name: 1435675577318_cluster
Type: Cluster
Size (#nodes): 3
Database(s): VMart
CPU Type: Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz
Vertica Version: 7.2.0

Buttons: API Key, View, Remove, Create Database

4. Follow the on-screen wizard, which prompts you to provide the following information:
 - Database name, which must be between 3–25 characters, starting with a letter, and followed by any combination of letters, numbers, or underscores.
 - (Optional) database administrator password for the database you want to create and connect to.
 - IP address of a node in your database cluster, typically the IP address of the administration host.
5. Click **Next**.

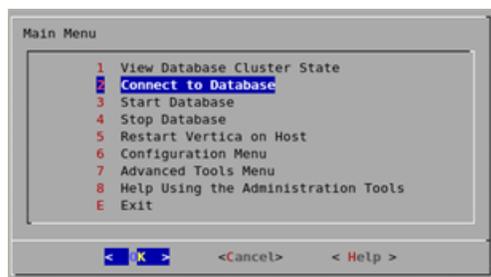
Step 3: Connecting to the Database

Regardless of the installation method you used, follow these steps to connect to the database.

1. As `dbadmin`, run the Administration Tools.

```
$ /opt/vertica/bin/admintools
```

or simply type `admintools`.
2. If you are already in the Administration Tools, navigate to the Main Menu page.
3. Select **Connect to Database**, click **OK**.



To configure and load data into the VMart database, complete the following steps:

- [Step 4: Defining the Database Schema](#)
- [Step 5: Loading Data](#)

If you installed the VMart database using the Quick Installation method, the schema, tables, and data are already defined. You can choose to drop the example database (see

[Restoring the Status of Your Host](#) in this guide) and perform the Advanced Installation, or continue straight to [Querying Your Data](#) in this guide.

Step 4: Defining the Database Schema

The VMart database installs with sample scripts with SQL commands that are intended to represent queries that might be used in a real business. The `vmart_define_schema.sql` script runs a script that defines the VMart schema and creates tables. You must run this script before you load data into the VMart database.

This script performs the following tasks:

- Defines two schemas in the VMart database schema: *online_sales* and *store*.
- Defines tables in both schemas.
- Defines constraints on those tables.

```
Vmart=> \i vmart_define_schema.sql
CREATE SCHEMA
CREATE SCHEMA
CREATE TABLE
ALTER TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE
```

Step 5: Loading Data

Now that you have created the schemas and tables, you can load data into a table by running the `vmart_load_data.sql` script. This script loads data from the 15 `.tbl` text files in `opt/vertica/examples/VMart_Schema` into the tables that `vmart_design_schema.sql` created.

It might take several minutes to load the data on a typical hardware cluster. Check the load status by monitoring the `vertica.log` file, as described in [Monitoring Log Files](#) in the Administrator's Guide.

```
VMart=> \i vmart_load_data.sql
Rows Loaded
-----
1826
(1 row)
Rows Loaded
-----
60000
(1 row)
Rows Loaded
-----
250
(1 row)
Rows Loaded
-----
1000
(1 row)
Rows Loaded
-----
50
(1 row)
Rows Loaded
-----
50000
(1 row)
Rows Loaded
-----
10000
(1 row)
Rows Loaded
-----
100
(1 row)
Rows Loaded
-----
100
(1 row)
Rows Loaded
-----
1000
(1 row)
Rows Loaded
-----
200
(1 row)
Rows Loaded
-----
```

```
5000000  
(1 row)  
  
Rows Loaded  
-----  
300000  
(1 row)  
VMart=>
```

Querying Data

The VMart database installs with sample scripts that contain SQL commands that represent queries that might be used in a real business. Use basic SQL commands to query the database, or try out the following command. Once you're comfortable running the example queries, you might want to write your own.

Note: The data that your queries return might differ from the example output shown in this guide because the sample data generator is random.

Type the following SQL command to return the values for five products with the lowest fat content in the Dairy department. The command selects the fat content from Dairy department products in the `product_dimension` table in the `public` schema, orders them from low to high and limits the output to the first five (the five lowest fat contents).

```
VMart => SELECT fat_content
        FROM ( SELECT DISTINCT fat_content
              FROM product_dimension
              WHERE department_description
                IN ('Dairy') ) AS food
        ORDER BY fat_content
        LIMIT 5;
```

Your results will be similar to the following:

```
fat_content
-----
80
81
82
83
84
(5 rows)
```

The preceding example is from the `vmart_query_01.sql` file. You can execute more sample queries using the scripts that installed with the VMart database or write your own. For a list of the sample queries supplied with Vertica, see the [Appendix](#).

Backing Up and Restoring the Database

Vertica supplies a comprehensive utility, the `vbr` Python script, that lets you back up and restore a full database, as well as create backups of specific schema or tables. The `vbr` utility creates backup directories during its initial execution; subsequently running the utility creates subdirectories.

The following information is intended to introduce the backup and restore functions. For more detailed information, see [Backing Up and Restoring the Database](#) in the *Administrator's Guide*.

Backing Up the Database

Use `vbr` to save your data to a variety of locations:

- A local directory on the nodes in a cluster
- One or more hosts outside of the cluster
- A different Vertica cluster (effectively cloning your database)

Note: Creating a database backup on a different cluster does not provide disaster recovery. The cloned database you create with `vbr` is entirely separate from the original, and is not kept synchronized with the database from which it is cloned.

When to Back up the Database

In addition to any guidelines established by your organization, Micro Focus recommends that you back up your database:

- Before you upgrade Vertica to another release.
- Before you drop a partition.
- After you load a large volume of data.
- If the epoch in the latest backup is earlier than the current ancient history mark (AHM).

- Before and after you add, remove, or replace nodes in your database cluster.
- After recovering a cluster from a crash.

Note: When you restore a database backup, you must restore to a cluster that is identical to the one where you created the backup. For this reason, always create a new backup after adding, removing, or replacing nodes.

Ideally, create regular backups of your full database. You can run the Vertica `vbr` utility from a cron job or other task scheduler.

Creating the Backup Configuration File

The `vbr` utility uses a configuration file for the information required to back up and restore a full- or object-level backup. The configuration file defines where the database backup is saved, the temporary directories it uses, and which nodes, schema, and/or tables in the database are to be backed up. You cannot run `vbr` without a configuration file, and no default file exists.

To invoke the script to set up a configuration file, enter this command:

```
$ vbr --setupconfig
```

The script prompts you to answer the following questions regarding the configuration file. Type **Enter** to accept the default value in parentheses. See [VBR Configuration File Reference](#) in the Administrator's Guide for information about specific questions.

```
[dbadmin@localhost ~]$ /opt/vertica/bin/vbr --setupconfig
Snapshot name (backup_snapshot): fullbak1
Number of restore points (1): 3
Specify objects (no default):
Object restore mode (coexist, createOrReplace or create)
(createOrReplace):
Vertica user name (dbadmin):
Save password to avoid runtime prompt? (n) [y/n]: y
Password to save in vbr config file (no default):
Node v_vmart_node0001
Backup host name (no default): 194.66.82.11
Backup directory (no default): /home/dbadmin/backups
Config file name (fullbak1.ini):
Password file name (no default value) (no default):pwdfile
Change advanced settings? (n) [y/n]: n
```

```
Saved vbr configuration to fullbak1.ini.  
Saved vbr database password to pwdfile.ini.
```

After you answer the required questions, vbr generates a configuration file with the information you supplied. Use the `Config file` name you specified when you run the `--task backup` or other commands. The `vbr` utility uses the configuration file contents for both backup and restore tasks.

Creating Full and Incremental Backups

Before you create a database backup, ensure the following:

- Your database is running.
- All of the backup hosts are up and available.
- The backup location host has sufficient disk space to store the backups.
- The user who starts the utility has write access to the target directories on the host backup location.

Run the `vbr` script from a terminal using the database administrator account from an initiator node in your database cluster. You cannot run the utility as root.

Use the `--task backup` and `--config-file filename` directives as shown in this example.

```
[release@qco55srv01:/scratch_b/qa/vertica/QA/VT_Scenario 0]$ vbr -t backup --config $FULLBAK_CONFIG  
Starting backup of database VTDB.  
Participating nodes: v_vmart_node0001, v_vmart_node0002, v_vmart_node0003, v_vmart_node0004.  
Snapshotting database.  
Snapshot complete.  
Approximate bytes to copy: 2315056043 of 2356089422 total.  
[=====] 100%  
Copying backup metadata.  
Finalizing backup.  
Backup complete!
```

By default, there is no screen output other than the progress bar.

If you do not specify a configuration file, the `vbr` utility searches for one at this location:

```
/opt/vertica/config/vbr.ini
```

If the utility does not locate the configuration you specify, it searches for one at `opt/vertica/config/vbr.ini`. If no file exists there, it fails with an error.

The first time you run the `vbr` utility, it performs a full backup; subsequent runs with the same configuration file create an incremental backup. When creating incremental backups, the utility copies new storage containers, which can include data that existed the last time you backed up the database, along with new and changed data since then. By default, `vbr` saves one archive backup, unless you set the `restorePointLimit` parameter value in the configuration file to a value greater than 1.

Restoring the Database

To restore a full database backup, ensure that:

- The database is down.
- All of the backup hosts are up and available.
- The backup directory exists and contains the backups from which to restore.

To begin a full database backup restore, log in using the database administrator's account. You cannot run the utility as root. For detailed instructions on restoring a database, refer to [Recovering the Database](#).

Using Database Designer to Create a Comprehensive Design

The Vertica Database Designer:

- Analyzes your logical schema, sample data, and, optionally, your sample queries.
- Creates a physical schema design (a set of projections) that can be deployed automatically or manually.
- Can be used by anyone without specialized database knowledge.
- Can be run and rerun any time for additional optimization without stopping the database.
- Uses strategies to provide optimal query performance and data compression.

Use Database Designer to create a comprehensive design, which allows you to create new projections for all tables in your database.

You can also use Database Designer to create an incremental design, which creates projections for all tables referenced in the queries you supply. For more information, see [Incremental Design](#) in the Administrator's Guide.

You can create a comprehensive design with Database Designer using Management Console or through Administration Tools. You can also choose to run Database Designer programmatically (See [About Running Database Designer Programmatically](#)).

This section shows you how to:

- [Running Database Designer with Management Console](#)
- [Run Database Designer with Administration Tools](#)

Running Database Designer with Management Console

In this tutorial, you'll create a comprehensive design with Database Designer through the Management Console interface. If, in the future, you have a query that you want to optimize, you can create an enhanced (incremental) design with additional projections. You can tune

these projections specifically for the query you provide. See [Comprehensive Design](#) in the Administrator's Guide for more information.

Note: To run Database Designer outside Administration Tools, you must be a dbadmin user. If you are not a dbadmin user, you must have the DBDUSER role assigned to you and own the tables for which you are designing projections.

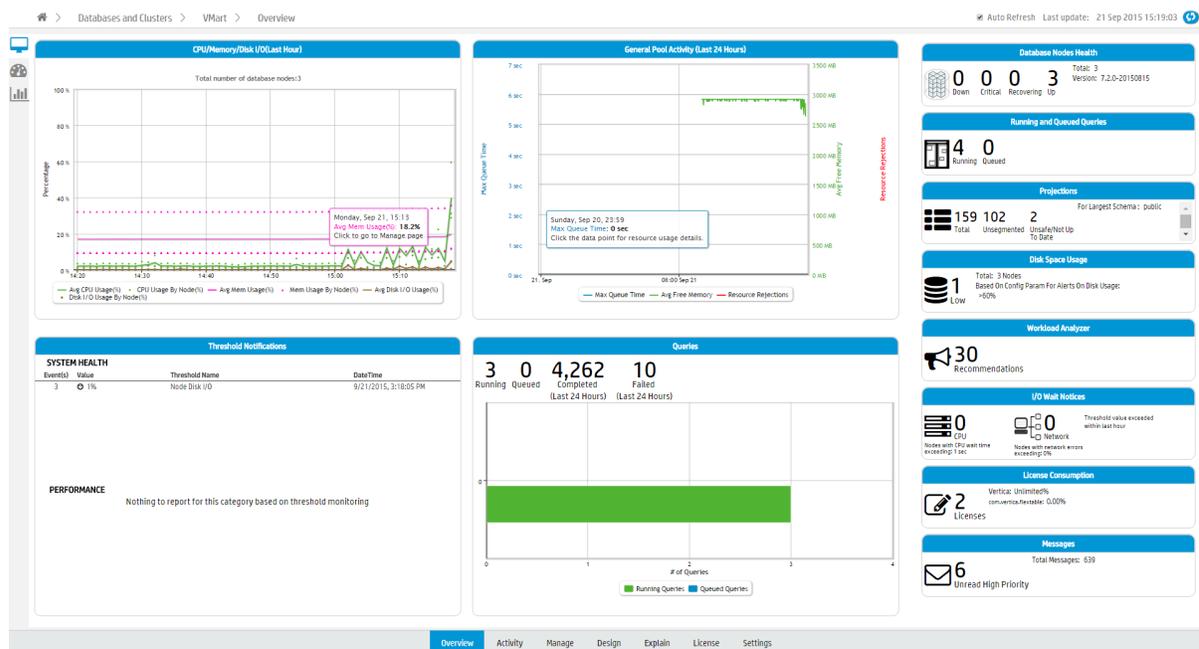
You can choose to create the design manually or use the wizard. To create a design manually, see [Creating a Design Manually](#) in the Administrator's Guide.

Set your browser so that it does not cache pages. If a browser caches pages, you may not be able to see the new design added.

Follow these steps to use the wizard to create the comprehensive design in Management Console:

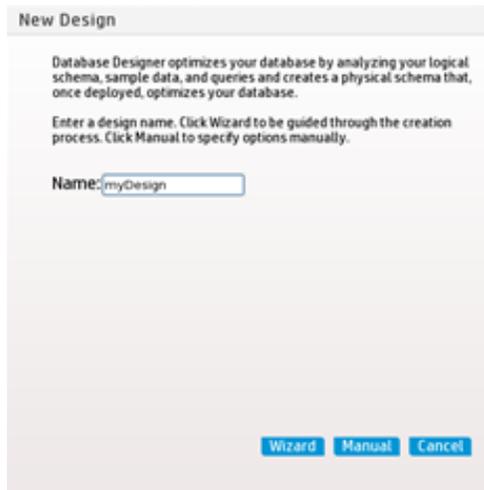
1. Log in to Management Console.
2. Verify that your database is up and running.
3. Choose the database for which you want to create the design. You can find the database under the **Recent Databases** section or by clicking **Existing Infrastructure** to reach the Databases and Clusters page.

The database overview page opens:

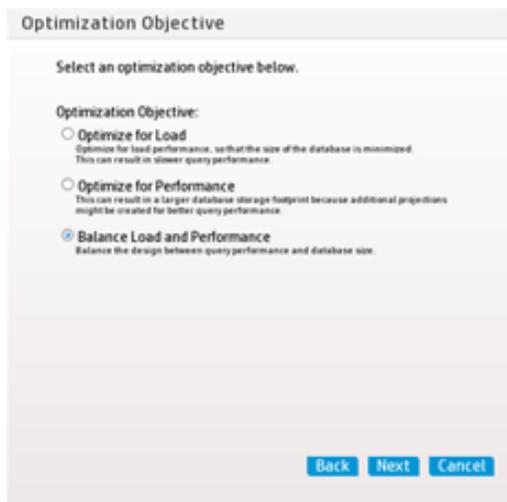


4. At the bottom of the screen, click the **Design** button.

5. In the **New Design** dialog box, enter the design name.



6. Click **Wizard** to continue.
7. Create an initial design. For **Design Type**, select **Comprehensive** and click **Next**.
8. In the **Optimization Objective** window, select **Balance Load and Performance** to create a design that is balanced between database size and query performance. Click **Next**.



9. Select the schemas. Because the VMart design is a multi-schema database, select all three schemas (public, store, and online_sales) for your design in the **Select Sample Data** window. Click **Next**.



If you include a schema that contains tables without data, the design could be suboptimal. You can choose to continue, but Vertica recommends that you deselect the schemas that contain empty tables before you proceed.

10. Choose the K-safety value for your design. The K-Safety value determines the number of buddy projections you want database designer to create.
11. Choose Analyze Correlations Mode. Analyze Correlations Mode determines if Database Designer analyzes and considers column correlations when creating the design.
 - **Ignore:** When creating a design, ignore any column correlations in the specified tables.
 - **Consider existing:** Consider the existing correlations in the tables when creating the design. If you set the mode to 1, and there are no existing correlations, Database Designer does not consider correlations.
 - **Analyze missing:** Analyze column correlations on tables where the correlation analysis was not previously performed. When creating the design, consider all column correlations (new and existing).
 - **Analyze all:** Analyze all column correlations in the tables and consider them when creating the design. Even if correlations exist for a table, reanalyze the table for correlations.

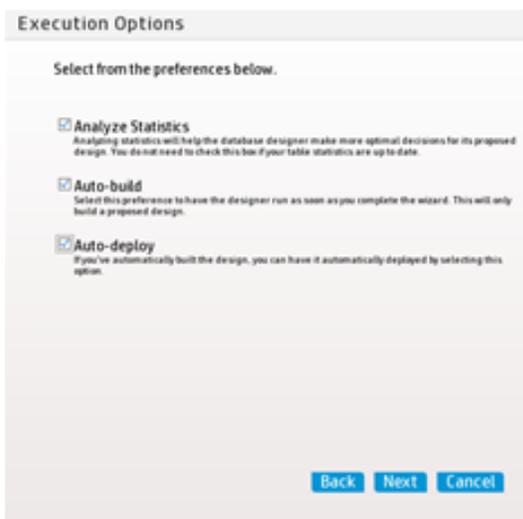
Click **Next**.

12. Submit query files to Database Designer in one of two ways:

- Supply your own query files by selecting the **Browse** button.
- Click **Use Query Repository**, which submits recently executed queries from the QUERY_REQUESTS system table.

Click **Next**.

13. In the **Execution Options** window, select all the options you want. You can select all three options or fewer.



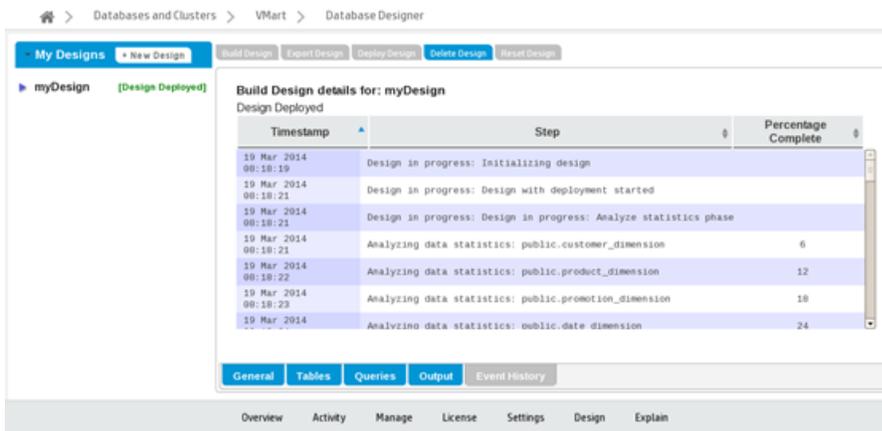
The three options are:

- **Analyze statistics:** Select this option to run statistics automatically after design deploy to help Database Designer make more optimal decisions for its proposed design.
- **Auto-build:** Select this option to run Database Designer as soon as you complete the wizard. This option only builds the proposed design.
- **Auto-deploy:** Select this option for auto-build designs that you want to deploy automatically.

14. Click **Submit Design**.

The **Database Designer** page opens:

- If you chose to automatically deploy your design, Database Designer executes in the background.
- If you did not select the **Auto-build** or **Auto-deploy** options, you can click **Build Design** or **Deploy Design** on the **Database Designer** page.



15. In the **My Designs** pane, view the status of your design:
 - When the deployment completes, the **My Design** pane shows **Design Deployed**.
 - The event history window shows the details of the design build and deployment.

To run Database Designer with Administration Tools, see [Run Database Designer with Administration Tools](#) in this guide.

Run Database Designer with Administration Tools

In this procedure, you create a comprehensive design with Database Designer using the Administration Tools interface. If, in the future, you have a query that you want to optimize, you can create an enhanced (incremental) design with additional projections. You can tune these projections specifically for the query you provide. See [Incremental Design](#) in the Administrator's Guide for more information.

Follow these steps to create the comprehensive design using Database Designer in Administration Tools:

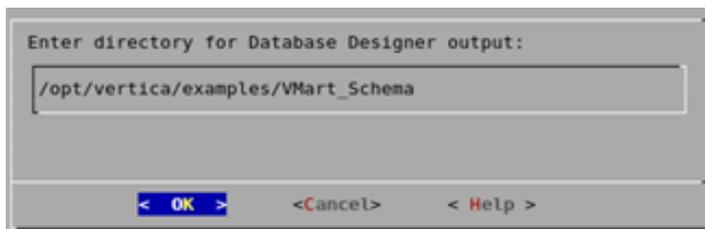
1. If you are not in Administration Tools, exit the vsql session and access Administration Tools:
 - Type `\q` to exit vsql.
 - Type `admintools` to access the Administration Tools Main Menu.

2. Start the database for which you want to create a design.
3. From the **Main Menu**, click **Configuration Menu** and then click **OK**.
4. From the **Configuration Menu**, click **Run Database Designer** and then click **OK**.
5. When the **Select a database for design** dialog box opens, select **VMart** and then click **OK**.

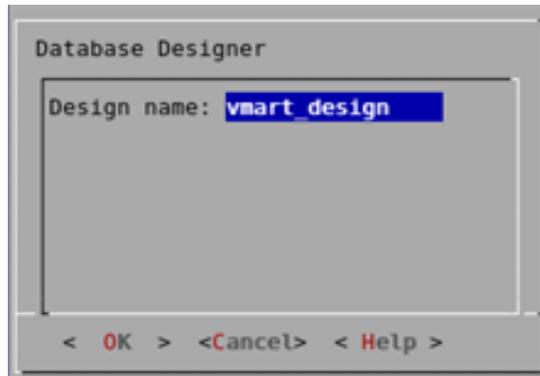


If you are prompted to enter the password for the database, click **OK** to bypass the message. Because no password was assigned when you installed the VMart database, you do not need to enter one now.

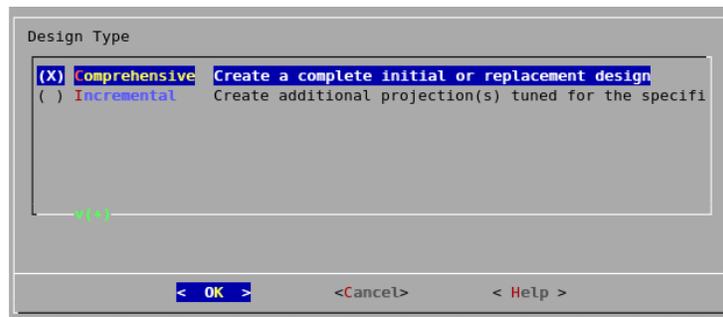
6. Click **OK** to accept the default directory for storing Database Designer output and log files.



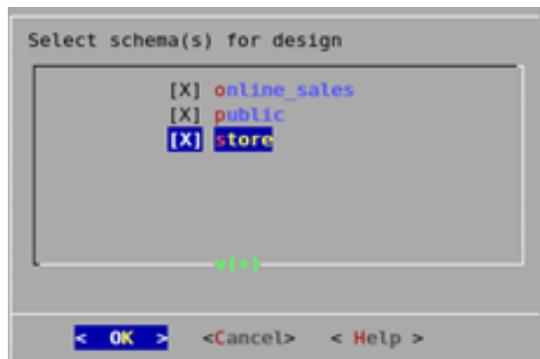
7. In the **Database Designer** window, enter a name for the design, for example, `vmart_design`, and click **OK**. Design names can contain only alphanumeric characters or underscores. No other special characters are allowed.



8. Create a complete initial design. In the **Design Type** window, click **Comprehensive** and click **OK**.

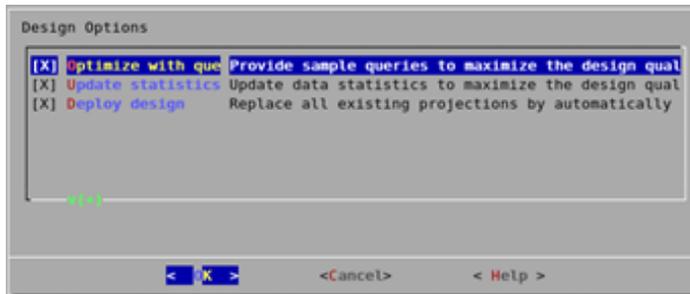


9. Select the schemas. Because the VMart design is a multi-schema database, you can select all three schemas (online_sales, public, and store) for your design. Click **OK**.



If you include a schema that contains tables without data, the Administration Tools notifies you that designing for tables without data could be suboptimal. You can choose to continue, but Micro Focus recommends that you deselect the schemas that contain empty tables before you proceed.

10. In the **Design Options** window, accept all three options and click **OK**.

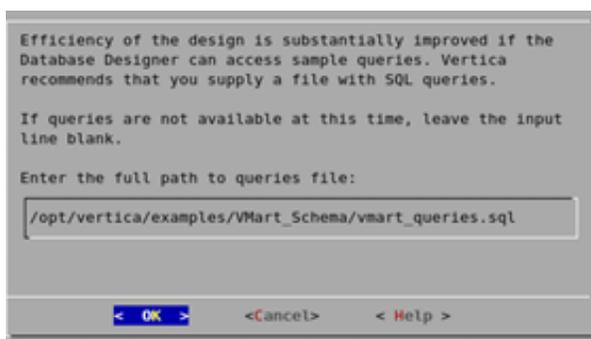


The three options are:

- **Optimize with queries:** Supplying the Database Designer with queries is especially important if you want to optimize the database design for query performance. Micro Focus recommends that you limit the design input to 100 queries.
- **Update statistics:** Accurate statistics help the Database Designer choose the best strategy for data compression. If you select this option, the database statistics are updated to maximize design quality.
- **Deploy design:** The new design deploys automatically. During deployment, new projections are added, some existing projections retained, and any necessary existing projections removed. Any new projections are refreshed to populate them with data.

11. Because you selected the **Optimize with queries** option, you must enter the full path to the file containing the queries that will be run on your database. In this example, it is:

```
/opt/vertica/examples/VMart_Schema/vmart_queries.sql
```

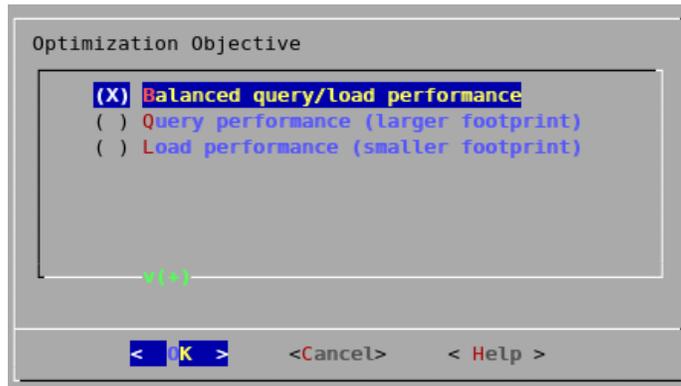


The queries in the query file must be delimited with a semicolon (;).

12. Choose the K-safety value you want and click **OK**. The design K-Safety determines the number of buddy projections you want database designer to create.

If you create a comprehensive design on a single node, you are not prompted to enter a K-safety value.

13. In the **Optimization Objective** window, select **Balanced query/load performance** to create a design that is balanced between database size and query performance. Click **OK**.



14. When the informational message displays, click **Proceed**.

Database Designer automatically performs these actions:

- Sets up the design session.
- Examines table data.
- Loads queries from the query file you provided (in this example, `/opt/vertica/examples/VMart_Schema/vmart_queries.sql`).
- Creates the design.

Deploys the design or saves a SQL file containing the commands to create the design, based on your selections in the Design Options window.

Depending on system resources, the design process could take several minutes. You should allow this process to complete uninterrupted. If you must cancel the session, use Ctrl+C.

```
Database Designer started.  
  
For large databases a design session could take a long time; allow it to complete uninterrupted.  
Use Ctrl+C if you must cancel the session.  
  
Setting up design session...  
  
Examining table data...  
  
Loading queries from '/opt/vertica/examples/VMart_Schema/vmart_queries.sql'.  
Processed 9 SQL statement(s), all accepted and considered in the design.  
No existing projections found.  
  
Creating design and deploying projections...
```

15. When Database Designer finishes, press **Enter** to return to the Administration Tools menu. Examine the steps taken to create the design. The files are in the directory you specified to store the output and log files. In this example, that directory is `/opt/vertica/examples/VMart_Schema`. For more information about the script files, see [About Database Designer](#), in the Administrator's Guide.

For additional information about managing your designs, see [Creating a Database Design](#) in the Administrator's Guide.

Restoring the Status of Your Host

When you finish the tutorial, you can restore your host machines to their original state. Use the following instructions to clean up your host and start over from scratch.

Stopping and Dropping the Database

Follow these steps to stop and/or drop your database. A database must be stopped before it can be dropped.

1. If connected to the database, disconnect by typing `\q`.
2. In the Administration Tools **Main Menu** dialog box, click **Stop Database** and click **OK**.
3. In the **Select database to stop** window, select the database you want to stop and click **OK**.
4. After stopping the database, click **Configuration Menu** and click **OK**.
5. Click **Drop Database** and click **OK**.
6. In the **Select database to drop** window, select the database you want to drop and click **OK**.
7. Click **Yes** to confirm.
8. In the next window type `yes` (lowercase) to confirm and click **OK**.

Alternatively, use the `delete_example` script, which stops and drops the database:

1. If connected to the database, disconnect by typing `\q`.
2. In the Administration Tools **Main Menu** dialog box, select **Exit**.
3. Log in as the database administrator.
4. Change to the `/examples` directory.

```
$ cd /opt/vertica/examples
```

5. Run the `delete_example` script.

```
$ /opt/vertica/sbin/delete_example Vmart
```

Uninstalling Vertica

See [Uninstalling Vertica](#).

Optional Steps

You can also choose to:

- Remove the `dbadmin` account on all cluster hosts.
- Remove any example database directories you created.

Changing the GUI Appearance

The appearance of the Graphical User Interface (GUI) depends on the color and font settings used by your terminal window. The screen captures in this document were made using the default color and font settings in a PuTTY terminal application running on a Windows platform.

Note: If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, make sure your window is at least 81 characters wide and 23 characters high.

If you are using PuTTY, take these steps to make the Administration Tools look like the screen captures in this document.

1. In a PuTTY window, right-click the title area and select **Change Settings**.
2. Create or load a saved session.
3. In the **Category** dialog, click **Window > Appearance**.
4. In the **Font** settings, click the **Change...** button.
5. Select **Font:** Courier New, **Regular Size:** 10.
6. Click **Apply**.

Repeat these steps for each existing session that you use to run the Administration Tools.

You can also change the translation to support UTF-8.

1. In a PuTTY window, right-click the title area and select **Change Settings**.
2. Create or load a saved session.
3. In the **Category** dialog, click **Window > Translation**.
4. In the **Received data assumed to be in which character set** drop-down menu, select **UTF-8**.
5. Click **Apply**.

Appendix: VMart Example Database Schema, Tables, and Scripts

The Appendix provides detailed information about the VMart example database's schema, tables, and scripts.

The VMart example database contains three different schemas:

- `public`
- `store`
- `online_sales`

The term “schema” has several related meanings in Vertica:

- In SQL statements, a schema refers to named namespace for a logical schema.
- Logical schema refers to a set of tables and constraints.
- Physical schema refers to a set of projections.

Each schema contains tables that are created and loaded during database installation. See the schema maps for a list of tables and their contents:

- [public Schema Map](#)
- [store Schema Map](#)
- [online_sales Schema Map](#)

The VMart database installs with sample scripts that contain SQL commands that represent queries that might be used in a real business. The sample scripts are available in the [Sample Scripts](#) section of this Appendix. Once you're comfortable running the example queries, you might want to write your own.

Tables

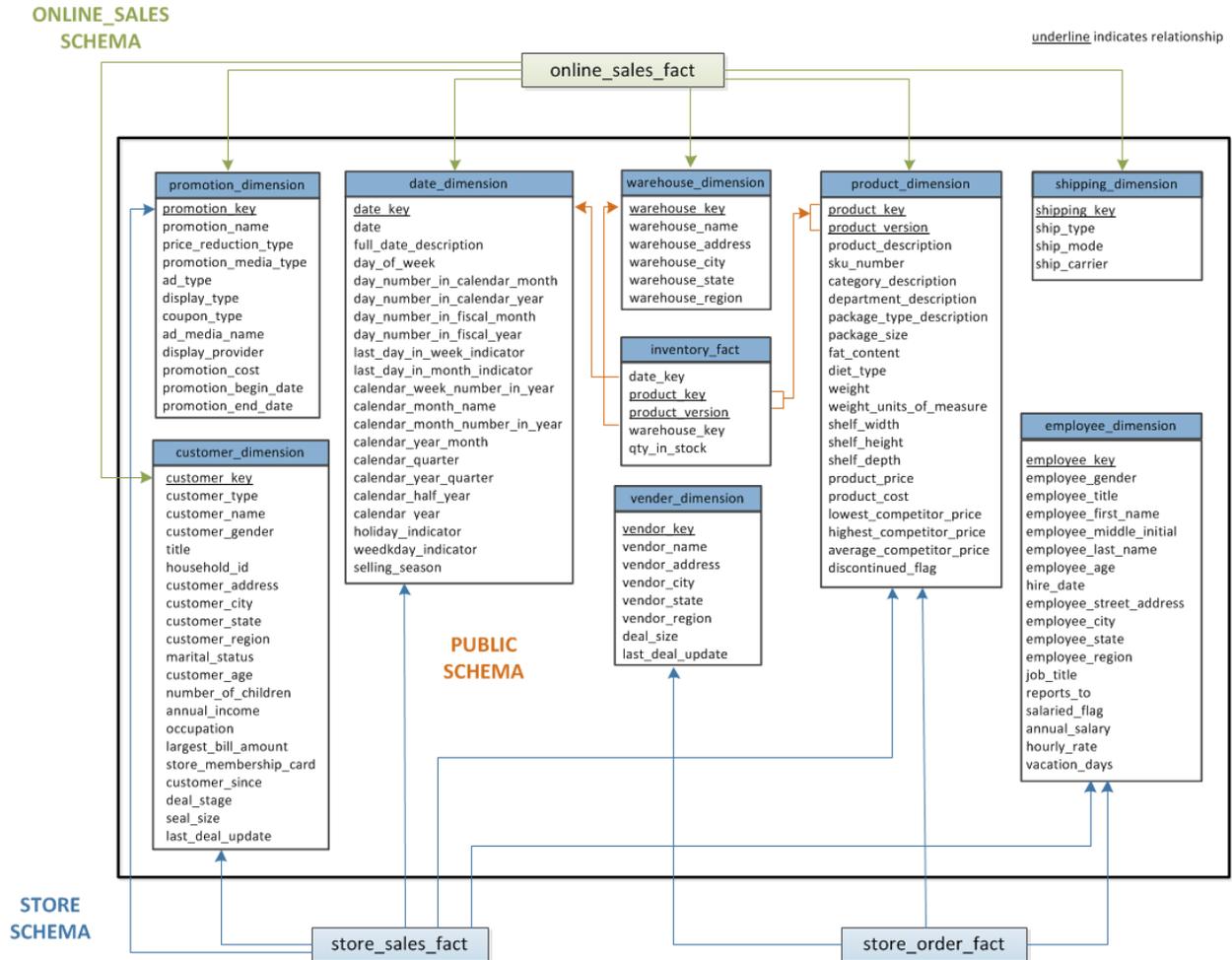
The three schemas in the VMart database include the following tables:

<code>public</code> Schema	<code>store</code> Schema	<code>online_sales</code>
----------------------------	---------------------------	---------------------------

		Schema
inventory_fact	store_orders_fact	online_sales_fact
customer_dimension	store_sales_fact	call_center_dimension
date_dimension	store_dimension	online_page_dimension
employee_dimension		
product_dimension		
promotion_dimension		
shipping_dimension		
vendor_dimension		
warehouse_dimension		

public Schema Map

The `public` schema is a snowflake schema. The following graphic illustrates the `public` schema and its relationships with tables in the `online_sales` and `store` schemas.



inventory_fact

This table contains information about each product in inventory.

Column Name	Data Type	NULLS
date_key	INTEGER	No
product_key	INTEGER	No
product_version	INTEGER	No
warehouse_key	INTEGER	No
qty_in_stock	INTEGER	No

customer_dimension

This table contains information about all the retail chain's customers.

Column Name	Data Type	NULLs
customer_key	INTEGER	No
customer_type	VARCHAR(16)	Yes
customer_name	VARCHAR(256)	Yes
customer_gender	VARCHAR(8)	Yes
title	VARCHAR(8)	Yes
household_id	INTEGER	Yes
customer_address	VARCHAR(256)	Yes
customer_city	VARCHAR(64)	Yes
customer_state	CHAR(2)	Yes
customer_region	VARCHAR(64)	Yes
marital_status	VARCHAR(32)	Yes
customer_age	INTEGER	Yes
number_of_children	INTEGER	Yes
annual_income	INTEGER	Yes
occupation	VARCHAR(64)	Yes
largest_bill_amount	INTEGER	Yes
store_membership_card	INTEGER	Yes
customer_since	DATE	Yes
deal_stage	VARCHAR(32)	Yes
deal_size	INTEGER	Yes

last_deal_update	DATE	Yes
------------------	------	-----

date_dimension

This table contains information about dates. It is generated from a file containing correct date/time data.

Column Name	Data Type	NULLs
date_key	INTEGER	No
date	DATE	Yes
full_date_description	VARCHAR(18)	Yes
day_of_week	VARCHAR(9)	Yes
day_number_in_calendar_month	INTEGER	Yes
day_number_in_calendar_year	INTEGER	Yes
day_number_in_fiscal_month	INTEGER	Yes
day_number_in_fiscal_year	INTEGER	Yes
last_day_in_week_indicator	INTEGER	Yes
last_day_in_month_indicator	INTEGER	Yes
calendar_week_number_in_year	INTEGER	Yes
calendar_month_name	VARCHAR(9)	Yes
calendar_month_number_in_year	INTEGER	Yes
calendar_year_month	CHAR(7)	Yes
calendar_quarter	INTEGER	Yes
calendar_year_quarter	CHAR(7)	Yes
calendar_half_year	INTEGER	Yes
calendar_year	INTEGER	Yes

holiday_indicator	VARCHAR(10)	Yes
weekday_indicator	CHAR(7)	Yes
selling_season	VARCHAR(32)	Yes

employee_dimension

This table contains information about all the people who work for the retail chain.

Column Name	Data Type	NULLs
employee_key	INTEGER	No
employee_gender	VARCHAR(8)	Yes
courtesy_title	VARCHAR(8)	Yes
employee_first_name	VARCHAR(64)	Yes
employee_middle_initial	VARCHAR(8)	Yes
employee_last_name	VARCHAR(64)	Yes
employee_age	INTEGER	Yes
hire_date	DATE	Yes
employee_street_address	VARCHAR(256)	Yes
employee_city	VARCHAR(64)	Yes
employee_state	CHAR(2)	Yes
employee_region	CHAR(32)	Yes
job_title	VARCHAR(64)	Yes
reports_to	INTEGER	Yes
salaried_flag	INTEGER	Yes
annual_salary	INTEGER	Yes
hourly_rate	FLOAT	Yes

vacation_days	INTEGER	Yes
---------------	---------	-----

product_dimension

This table describes all products sold by the department store chain.

Column Name	Data Type	NULLs
product_key	INTEGER	No
product_version	INTEGER	No
product_description	VARCHAR(128)	Yes
sku_number	CHAR(32)	Yes
category_description	CHAR(32)	Yes
department_description	CHAR(32)	Yes
package_type_description	CHAR(32)	Yes
package_size	CHAR(32)	Yes
fat_content	INTEGER	Yes
diet_type	CHAR(32)	Yes
weight	INTEGER	Yes
weight_units_of_measure	CHAR(32)	Yes
shelf_width	INTEGER	Yes
shelf_height	INTEGER	Yes
shelf_depth	INTEGER	Yes
product_price	INTEGER	Yes
product_cost	INTEGER	Yes
lowest_competitor_price	INTEGER	Yes
highest_competitor_price	INTEGER	Yes

average_competitor_price	INTEGER	Yes
discontinued_flag	INTEGER	Yes

promotion_dimension

This table describes every promotion ever done by the retail chain.

Column Name	Data Type	NULLs
promotion_key	INTEGER	No
promotion_name	VARCHAR(128)	Yes
price_reduction_type	VARCHAR(32)	Yes
promotion_media_type	VARCHAR(32)	Yes
ad_type	VARCHAR(32)	Yes
display_type	VARCHAR(32)	Yes
coupon_type	VARCHAR(32)	Yes
ad_media_name	VARCHAR(32)	Yes
display_provider	VARCHAR(128)	Yes
promotion_cost	INTEGER	Yes
promotion_begin_date	DATE	Yes
promotion_end_date	DATE	Yes

shipping_dimension

This table contains information about shipping companies that the retail chain uses.

Column Name	Data Type	NULLs
shipping_key	INTEGER	No

ship_type	CHAR(30)	Yes
ship_mode	CHAR(10)	Yes
ship_carrier	CHAR(20)	Yes

vendor_dimension

This table contains information about each vendor that provides products sold through the retail chain.

Column Name	Data Type	NULLs
vendor_key	INTEGER	No
vendor_name	VARCHAR(64)	Yes
vendor_address	VARCHAR(64)	Yes
vendor_city	VARCHAR(64)	Yes
vendor_state	CHAR(2)	Yes
vendor_region	VARCHAR(32)	Yes
deal_size	INTEGER	Yes
last_deal_update	DATE	Yes

warehouse_dimension

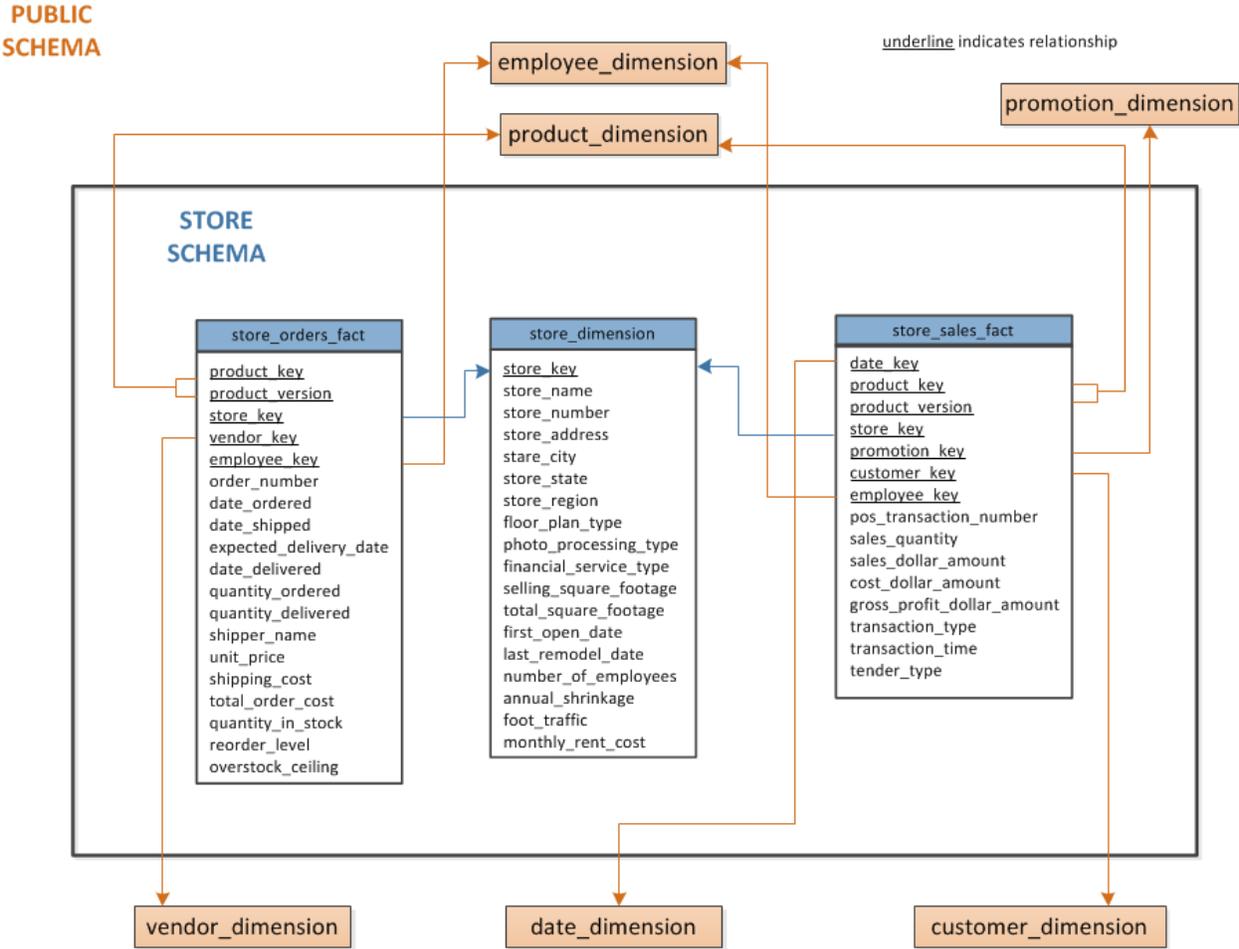
This table provides information about each of the chain's warehouses.

Column Name	Data Type	NULLs
warehouse_key	INTEGER	No
warehouse_name	VARCHAR(20)	Yes
warehouse_address	VARCHAR(256)	Yes
warehouse_city	VARCHAR(60)	Yes

warehouse_state	CHAR(2)	Yes
warehouse_region	VARCHAR(32)	Yes

store Schema Map

The store schema is a snowflake schema that contains information about the retail chain’s bricks-and-mortar stores. The following graphic illustrates the store schema and its relationship with tables in the public schema.



store_orders_fact

This table contains information about all orders made at the company's brick-and-mortar stores.

Column Name	Data Type	NULLs
product_key	INTEGER	No
product_version	INTEGER	No
store_key	INTEGER	No
vendor_key	INTEGER	No
employee_key	INTEGER	No
order_number	INTEGER	No
date_ordered	DATE	Yes
date_shipped	DATE	Yes
expected_delivery_date	DATE	Yes
date_delivered	DATE	Yes
quantity_ordered	INTEGER	Yes
quantity_delivered	INTEGER	Yes
shipper_name	VARCHAR(32)	Yes
unit_price	INTEGER	Yes
shipping_cost	INTEGER	Yes
total_order_cost	INTEGER	Yes
quantity_in_stock	INTEGER	Yes
reorder_level	INTEGER	Yes
overstock_ceiling	INTEGER	Yes

store_sales_fact

This table contains information about all sales made at the company's brick-and-mortar stores.

Column Name	Data Type	NULLs
date_key	INTEGER	No
product_key	INTEGER	No
product_version	INTEGER	No
store_key	INTEGER	No
promotion_key	INTEGER	No
customer_key	INTEGER	No
employee_key	INTEGER	No
pos_transaction_number	INTEGER	No
sales_quantity	INTEGER	Yes
sales_dollar_amount	INTEGER	Yes
cost_dollar_amount	INTEGER	Yes
gross_profit_dollar_amount	INTEGER	Yes
transaction_type	VARCHAR(16)	Yes
transaction_time	TIME	Yes
tender_type	VARCHAR(8)	Yes

store_dimension

This table contains information about each brick-and-mortar store within the retail chain.

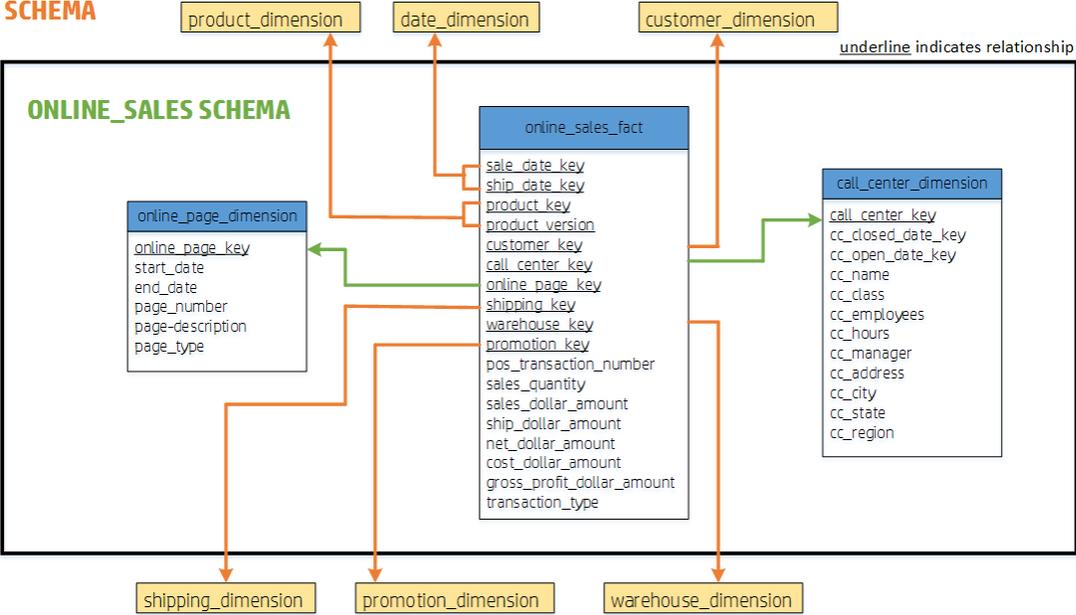
Column Name	Data Type	NULLs
store_key	INTEGER	No

store_name	VARCHAR(64)	Yes
store_number	INTEGER	Yes
store_address	VARCHAR(256)	Yes
store_city	VARCHAR(64)	Yes
store_state	CHAR(2)	Yes
store_region	VARCHAR(64)	Yes
floor_plan_type	VARCHAR(32)	Yes
photo_processing_type	VARCHAR(32)	Yes
financial_service_type	VARCHAR(32)	Yes
selling_square_footage	INTEGER	Yes
total_square_footage	INTEGER	Yes
first_open_date	DATE	Yes
last_remodel_date	DATE	Yes
number_of_employees	INTEGER	Yes
annual_shrinkage	INTEGER	Yes
foot_traffic	INTEGER	Yes
monthly_rent_cost	INTEGER	Yes

online_sales Schema Map

The `online_sales` schema is a snowflake schema that contains information about the retail chains. The following graphic illustrates the `online_sales` schema and its relationship with tables in the `public` schema.

PUBLIC SCHEMA



online_sales_fact

This table describes all the items purchased through the online store front.

Column Name	Data Type	NULLs
sale_date_key	INTEGER	No
ship_date_key	INTEGER	No
product_key	INTEGER	No
product_version	INTEGER	No
customer_key	INTEGER	No
call_center_key	INTEGER	No
online_page_key	INTEGER	No
shipping_key	INTEGER	No
warehouse_key	INTEGER	No
promotion_key	INTEGER	No

pos_transaction_number	INTEGER	No
sales_quantity	INTEGER	Yes
sales_dollar_amount	FLOAT	Yes
ship_dollar_amount	FLOAT	Yes
net_dollar_amount	FLOAT	Yes
cost_dollar_amount	FLOAT	Yes
gross_profit_dollar_amount	FLOAT	Yes
transaction_type	VARCHAR(16)	Yes

call_center_dimension

This table describes all the chain's call centers.

Column Name	Data Type	NULLs
call_center_key	INTEGER	No
cc_closed_date	DATE	Yes
cc_open_date	DATE	Yes
cc_date	VARCHAR(50)	Yes
cc_class	VARCHAR(50)	Yes
cc_employees	INTEGER	Yes
cc_hours	CHAR(20)	Yes
cc_manager	VARCHAR(40)	Yes
cc_address	VARCHAR(256)	Yes
cc_city	VARCHAR(64)	Yes
cc_state	CHAR(2)	Yes
cc_region	VARCHAR(64)	Yes

online_page_dimension

This table describes all the pages in the online store front.

Column Name	Data Type	NULLs
online_page_key	INTEGER	No
start_date	DATE	Yes
end_date	DATE	Yes
page_number	INTEGER	Yes
page_description	VARCHAR(100)	Yes
page_type	VARCHAR(100)	Yes

Sample Scripts

You can create your own queries, but the VMart example directory includes sample query script files to help you get started quickly.

You can find the following sample scripts at this path `/opt/vertica/examples/VMart_Schema`.

To run any of the scripts, enter

```
=> \i <script_name>
```

Alternatively, type the commands from the script file manually.

Note: The data that your queries return might differ from the example output shown in this guide because the sample data generator is random.

vmart_query_01.sql

```
-- vmart_query_01.sql  
-- FROM clause subquery
```

```
-- Return the values for five products with the
-- lowest-fat content in the Dairy department

SELECT fat_content
FROM (
  SELECT DISTINCT fat_content
  FROM product_dimension
  WHERE department_description
  IN ('Dairy') ) AS food
ORDER BY fat_content
LIMIT 5;
```

Output

```
fat_content
-----
      80
      81
      82
      83
      84
(5 rows)
```

vmart_query_02.sql

```
-- vmart_query_02.sql
-- WHERE clause subquery
-- Asks for all orders placed by stores located in Massachusetts
-- and by vendors located elsewhere before March 1, 2003:

SELECT order_number, date_ordered
FROM store.store_orders_fact orders
WHERE orders.store_key IN (
  SELECT store_key
  FROM store.store_dimension
  WHERE store_state = 'MA')
  AND orders.vendor_key NOT IN (
  SELECT vendor_key
  FROM public.vendor_dimension
  WHERE vendor_state = 'MA')
  AND date_ordered < '2012-03-01';
```

Output

```
order_number | date_ordered
-----+-----
      53019 | 2012-02-10
      222168 | 2012-02-05
      160801 | 2012-01-08
      106922 | 2012-02-07
      246465 | 2012-02-10
      234218 | 2012-02-03
      263119 | 2012-01-04
       73015 | 2012-01-01
```

```
233618 | 2012-02-10
85784 | 2012-02-07
146607 | 2012-02-07
296193 | 2012-02-05
55052 | 2012-01-05
144574 | 2012-01-05
117412 | 2012-02-08
276288 | 2012-02-08
185103 | 2012-01-03
282274 | 2012-01-01
245300 | 2012-02-06
143526 | 2012-01-04
59564 | 2012-02-06
...
```

vmart_query_03.sql

```
-- vmart_query_03.sql
-- noncorrelated subquery
-- Requests female and male customers with the maximum
-- annual income from customers

SELECT customer_name, annual_income
FROM public.customer_dimension
WHERE (customer_gender, annual_income) IN (
  SELECT customer_gender, MAX(annual_income)
  FROM public.customer_dimension
  GROUP BY customer_gender);
```

Output

```
customer_name | annual_income
-----+-----
James M. McNulty |          999979
Emily G. Vogel |          999998
(2 rows)
```

vmart_query_04.sql

```
-- vmart_query_04.sql
-- IN predicate
-- Find all products supplied by stores in MA

SELECT DISTINCT s.product_key, p.product_description
FROM store.store_sales_fact s, public.product_dimension p
WHERE s.product_key = p.product_key
AND s.product_version = p.product_version AND s.store_key IN (
  SELECT store_key
  FROM store.store_dimension
```

```
WHERE store_state = 'MA')  
ORDER BY s.product_key;
```

Output

product_key	product_description
1	Brand #1 butter
1	Brand #2 bagels
2	Brand #3 lamb
2	Brand #4 brandy
2	Brand #5 golf clubs
2	Brand #6 chicken noodle soup
3	Brand #10 ground beef
3	Brand #11 vanilla ice cream
3	Brand #7 canned chicken broth
3	Brand #8 halibut
3	Brand #9 camera case
4	Brand #12 rash ointment
4	Brand #13 low fat milk
4	Brand #14 chocolate chip cookies
4	Brand #15 silver polishing cream
5	Brand #16 cod
5	Brand #17 band aids
6	Brand #18 bananas
6	Brand #19 starch
6	Brand #20 vegetable soup
6	Brand #21 bourbon
...	

vmart_query_05.sql

```
-- vmart_query_05.sql  
-- EXISTS predicate  
-- Get a list of all the orders placed by all stores on  
-- January 2, 2003 for the vendors with records in the  
-- vendor_dimension table  
  
SELECT store_key, order_number, date_ordered  
FROM store.store_orders_fact  
WHERE EXISTS (  
  SELECT 1  
  FROM public.vendor_dimension  
  WHERE public.vendor_dimension.vendor_key = store.store_orders_fact.vendor_key)  
  AND date_ordered = '2012-01-02';
```

Output

store_key	order_number	date_ordered
98	151837	2012-01-02
123	238372	2012-01-02
242	263973	2012-01-02

```
150 |      226047 | 2012-01-02
247 |      232273 | 2012-01-02
203 |      171649 | 2012-01-02
129 |       98723 | 2012-01-02
 80 |     265660 | 2012-01-02
231 |     271085 | 2012-01-02
149 |      12169 | 2012-01-02
141 |     201153 | 2012-01-02
  1 |      23715 | 2012-01-02
156 |      98182 | 2012-01-02
 44 |     229465 | 2012-01-02
178 |     141869 | 2012-01-02
134 |     44410 | 2012-01-02
141 |     129839 | 2012-01-02
205 |     54138 | 2012-01-02
113 |     63358 | 2012-01-02
 99 |     50142 | 2012-01-02
 44 |     131255 | 2012-01-02
```

...

vmart_query_06.sql

```
-- vmart_query_06.sql
-- EXISTS predicate
-- Orders placed by the vendor who got the best deal
-- on January 4, 2004

SELECT store_key, order_number, date_ordered
FROM store.store_orders_fact ord, public.vendor_dimension vd
WHERE ord.vendor_key = vd.vendor_key
AND vd.deal_size IN (
  SELECT MAX(deal_size)
  FROM public.vendor_dimension)
AND date_ordered = '2013-01-04';
```

Output

```
store_key | order_number | date_ordered
-----+-----+-----
 45 |      202416 | 2013-01-04
 24 |      250295 | 2013-01-04
121 |      251417 | 2013-01-04
198 |       75716 | 2013-01-04
166 |      36008 | 2013-01-04
 27 |      150241 | 2013-01-04
148 |      182207 | 2013-01-04
  9 |      188567 | 2013-01-04
113 |       66017 | 2013-01-04
```

...

vmart_query_07.sql

```
-- vmart_query_07.sql
-- Multicolumn subquery
-- Which products have the highest cost,
-- grouped by category and department

SELECT product_description, sku_number, department_description
FROM public.product_dimension
WHERE (category_description, department_description, product_cost) IN (
  SELECT category_description, department_description,
  MAX(product_cost) FROM product_dimension
  GROUP BY category_description, department_description);
```

Output

product_description	sku_number	department_description
Brand #601 steak	SKU-#601	Meat
Brand #649 brooms	SKU-#649	Cleaning supplies
Brand #677 veal	SKU-#677	Meat
Brand #1371 memory card	SKU-#1371	Photography
Brand #1761 catfish	SKU-#1761	Seafood
Brand #1810 frozen pizza	SKU-#1810	Frozen Goods
Brand #1979 canned peaches	SKU-#1979	Canned Goods
Brand #2097 apples	SKU-#2097	Produce
Brand #2287 lens cap	SKU-#2287	Photography
...		

vmart_query_08.sql

```
-- vmart_query_08.sql
-- Using pre-join projections to answer subqueries
-- between online_sales_fact and online_page_dimension

SELECT page_description, page_type, start_date, end_date
FROM online_sales.online_sales_fact f, online_sales.online_page_dimension d
WHERE f.online_page_key = d.online_page_key
AND page_number IN
  (SELECT MAX(page_number)
   FROM online_sales.online_page_dimension)
AND page_type = 'monthly' AND start_date = '2012-06-02';
```

Output

page_description	page_type	start_date	end_date
Online Page Description #1	monthly	2012-06-02	2012-06-11
Online Page Description #1	monthly	2012-06-02	2012-06-11
Online Page Description #1	monthly	2012-06-02	2012-06-11
Online Page Description #1	monthly	2012-06-02	2012-06-11

```
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
Online Page Description #1 | monthly | 2012-06-02 | 2012-06-11
(12 rows)
```

vmart_query_09.sql

```
-- vmart_query_09.sql
-- Equi join
-- Joins online_sales_fact table and the call_center_dimension
-- table with the ON clause

SELECT sales_quantity, sales_dollar_amount, transaction_type, cc_name
FROM online_sales.online_sales_fact
INNER JOIN online_sales.call_center_dimension
ON (online_sales.online_sales_fact.call_center_key
    = online_sales.call_center_dimension.call_center_key
    AND sale_date_key = 156)
ORDER BY sales_dollar_amount DESC;
```

Output

sales_quantity	sales_dollar_amount	transaction_type	cc_name
7	589	purchase	Central Midwest
8	589	purchase	South Midwest
8	589	purchase	California
1	587	purchase	New England
1	586	purchase	Other
1	584	purchase	New England
4	584	purchase	New England
7	581	purchase	Mid Atlantic
5	579	purchase	North Midwest
8	577	purchase	North Midwest
4	577	purchase	Central Midwest
2	575	purchase	Hawaii/Alaska
4	573	purchase	NY Metro
4	572	purchase	Central Midwest
1	570	purchase	Mid Atlantic
9	569	purchase	Southeastern
1	569	purchase	NY Metro
5	567	purchase	Other
7	567	purchase	Hawaii/Alaska
9	567	purchase	South Midwest
1	566	purchase	New England
...			

Administrator's Guide

Welcome to the Vertica Administrator's Guide. This document describes how to set up and maintain an Vertica Analytics Platform database.

Prerequisites

This document assumes that you have already:

- Become familiar with the concepts discussed in the [Vertica Concepts](#).
- Performed the procedures described in the [Installing Vertica](#):
 - Constructed a hardware platform.
 - Installed Linux.
 - Installed Vertica (configured a cluster of hosts).
- Followed the [Tutorial](#) in Getting Started to experiment with setting up an example database.

Administration Overview

This document describes the functions performed by an Vertica database administrator (DBA). Perform these tasks using only the dedicated database administrator account that was created when you installed Vertica. The examples in this documentation set assume that the administrative account name is dbadmin.

- To perform certain cluster configuration and administration tasks, the DBA (users of the administrative account) must be able to supply the root password for those hosts. If this requirement conflicts with your organization's security policies, these functions must be performed by your IT staff.
- If you perform administrative functions using a different account from the account provided during installation, Vertica encounters file ownership problems.
- If you share the administrative account password, make sure that only one user runs the Administration Tools at any time. Otherwise, automatic configuration propagation does not work correctly.
- The Administration Tools require that the calling user's shell be `/bin/bash`. Other shells give unexpected results and are not supported.

Managing Licenses

You must license Vertica in order to use it. Micro Focus supplies your license in the form of one or more license files, which encode the terms of your license. The licenses that are available include:

- `vlicense.dat`, for columnar tables.
- `vlicense_565_bytes.dat`, for data stored in a Hadoop environment with Vertica for SQL on Apache Hadoop.

To prevent introducing special characters that invalidate the license, do not open the license files in an editor or email client. Opening the file in this way can introduce special characters, such as line endings and file terminators, that may not be visible within the editor. Whether visible or not, these characters invalidate the license.

Applying License Files

For ease of Vertica Premium Edition and SQL on Hadoop installation, Micro Focus recommends that you copy the license file to `/tmp/vlicense.dat` on the Administration host.

Be careful not to change the license key file in any way when copying the file between Windows and Linux, or to any other location. To help prevent applications from trying to alter the file, enclose the license file in an archive file (such as a `.zip` or `.tar` file).

After copying the license file from one location to another, check that the copied file size is identical to that of the one you received from Vertica.

Obtaining a License Key File

To obtain a license key (for example, for Premium Edition or SQL on Hadoop), contact Vertica at: <http://www.vertica.com/about/contact-us/>

Your Vertica Community Edition download package includes the Community Edition license, which allows three nodes and 1TB of data. The Vertica Community Edition license does not expire.

Understanding Vertica Licenses

Vertica has flexible licensing terms. It can be licensed on the following bases:

- Term-based (valid until a specific date)
- Size-based (valid to store up to a specified amount of raw data)
- Both term- and size-based
- Unlimited duration and data storage
- Node-based with an unlimited number of CPUs and users (one node is a server acting as a single computer system, whether physical or virtual)

Vertica Community Edition licenses include 1 terabyte data and a limit of 3 nodes.

Vertica for SQL on Apache Hadoop is a separate product with its own license. This documentation covers both products. Consult your license agreement for details about available features and limitations.

Your license key has your licensing bases encoded into it. If you are unsure of your current license, you can [view your license information from within Vertica](#).

Vertica Analytics Platform License Types

Vertica Analytics Platform is a full-featured offering with all analytical functions described in this documentation. It is best used for advanced analytics and enterprise data warehousing. There are two editions, Community Edition and Premium Edition.

Vertica Community Edition: You can download and start using Community Edition for free. The Community Edition license allows customers the following:

- 3 node limit
- 1 terabyte data limit

Community Edition licenses cannot be installed co-located in a Hadoop infrastructure and used to query data stored in Hadoop formats.

Vertica Premium Edition: You can purchase the Premium Edition license. The Premium Edition license entitles customers to:

- No node limit
- Data amount as specified by the license
- Store data in HDFS and read limited data from HDFS, as specified in your license

Premium Edition licenses cannot be installed co-located in a Hadoop infrastructure.

Note: Vertica does not support license downgrades.

Vertica for SQL on Apache Hadoop License

Vertica for SQL on Apache Hadoop is a license for running Vertica on a Hadoop environment. This allows users to run Vertica on data that is in a shared storage environment. It is best used for exploring data in a Hadoop data lake. It can be used only in co-located Hadoop environments to query data stored in Hadoop (Hortonworks, MapR, or Cloudera).

Customers can purchase this term-based SQL on Apache Hadoop license per the number of nodes they plan to use in their Hadoop environment. The license then audits the number of nodes being used for compliance.

Installing or Upgrading a License Key

The steps you follow to apply your Vertica license key vary, depending on the type of license you are applying and whether you are upgrading your license. This section describes the following:

- [New Vertica License Installations](#)
- [Vertica License Renewals or Upgrades](#)

New Vertica License Installations

1. Copy the license key file to your Administration Host.
2. Ensure the license key's file permissions are set to 400 (read permissions).

3. Install Vertica as described in the Installing Vertica if you have not already done so. The interface prompts you for the license key file.
4. To install Community Edition, leave the default path blank and click **OK**. To apply your evaluation or Premium Edition license, enter the absolute path of the license key file you downloaded to your Administration Host and press **OK**. The first time you log in as the Database Administrator and run the Administration Tools, the interface prompts you to accept the End-User License Agreement (EULA).

Note: If you installed Management Console, the MC administrator can point to the location of the license key during Management Console configuration.

5. Choose **View EULA**.
6. Exit the EULA and choose **Accept EULA** to officially accept the EULA and continue installing the license, or choose **Reject EULA** to reject the EULA and return to the Advanced Menu.

Vertica License Renewals or Upgrades

If your license is expiring or you want your database to grow beyond your licensed data size, you must renew or upgrade your license. Once you have obtained your renewal or upgraded license key file, you can install it using Administration Tools or Management Console.

Uploading or Upgrading a License Key Using Administration Tools

1. Copy the license key file to your Administration Host.
2. Ensure the license key's file permissions are set to 400 (read permissions).
3. Start your database, if it is not already running.
4. In the Administration Tools, select **Advanced > Upgrade License Key** and click **OK**.
5. Enter the absolute path to your new license key file and click **OK**. The interface prompts you to accept the End-User License Agreement (EULA).
6. Choose **View EULA**.

7. Exit the EULA and choose Accept EULA to officially accept the EULA and continue installing the license, or choose Reject EULA to reject the EULA and return to the Advanced Tools menu.

Uploading or Upgrading a License Key Using Management Console

1. From your database's Overview page in Management Console, click the License tab. The License page displays. You can view your installed licenses on this page.
2. Click Install New License at the top of the License page.
3. Browse to the location of the license key from your local computer and upload the file.
4. Click Apply at the top of the page. Management Console prompts you to accept the End-User License Agreement (EULA).
5. Select the check box to officially accept the EULA and continue installing the license, or click Cancel to exit.

Note: As soon as you renew or upgrade your license key from either your Administration Host or Management Console, Vertica applies the license update. No further warnings appear.

Viewing Your License Status

You can use several functions to display your license terms and current status.

Examining Your License Key

Use the [DISPLAY_LICENSE](#) SQL function described in the SQL Reference Manual to display the license information. This function displays the dates for which your license is valid (or Perpetual if your license does not expire) and any raw data allowance. For example:

```
=> SELECT DISPLAY_LICENSE();
          DISPLAY_LICENSE
-----
Vertica Systems, Inc.
```

```
1/1/2011  
12/31/2011  
30  
50TB  
(1 row)
```

You can also query the [LICENSES](#) system table to view information about your installed licenses. This table displays your license types, the dates for which your licenses are valid, and the size and node limits your licenses impose.

Alternatively, use the [LICENSES](#) table in Management Console. On your database Overview page, click the License tab to view information about your installed licenses.

Viewing Your License Compliance

If your license includes a raw data size allowance, Vertica periodically audits your database's size to ensure it remains compliant with the license agreement. If your license has a term limit, Vertica also periodically checks to see if the license has expired. You can see the result of the latest audits using the [GET_COMPLIANCE_STATUS](#) function.

```
=> select GET_COMPLIANCE_STATUS();  
          GET_COMPLIANCE_STATUS  
  
-----  
Raw Data Size: 2.00GB +/- 0.003GB  
License Size : 4.000GB  
Utilization  : 50%  
Audit Time   : 2011-03-09 09:54:09.538704+00  
Compliance Status : The database is in compliance with respect to raw data size.  
License End Date: 04/06/2011  
Days Remaining: 28.59  
(1 row)
```

To see how your ORC/Parquet data is affecting your license compliance, see [Viewing License Compliance for Hadoop File Formats](#).

Viewing Your License Status Through MC

Information about license usage is on the Settings page. See [Monitoring Database Size for License Compliance](#).

Viewing License Compliance for Hadoop File Formats

You can use the [EXTERNAL_TABLE_DETAILS](#) system table to gather information about your use of Hadoop file formats. This information can help you assess your compliance with your Vertica license.

Vertica computes the values in this table at query time, so to avoid performance problems, restrict your queries to filter by `table_schema`, `table_name`, or `source_format`. These three columns are the only columns you can use in a predicate, but you may use all of the usual predicate operators.

```
=> SELECT * FROM EXTERNAL_TABLE_DETAILS
      WHERE source_format = 'PARQUET' OR source_format = 'ORC';
-[ RECORD 1 ]-----+-----
-----
schema_oid      | 45035996273704978
table_schema    | public
table_oid       | 45035996273760390
table_name      | ORC_demo
source_format   | ORC
total_file_count | 5
total_file_size_bytes | 789
source_statement | COPY FROM 'ORC_demo/*' ORC
file_access_error |
-[ RECORD 2 ]-----+-----
-----
schema_oid      | 45035196277204374
table_schema    | public
table_oid       | 45035996274460352
table_name      | Parquet_demo
source_format   | PARQUET
total_file_count | 3
total_file_size_bytes | 498
source_statement | COPY FROM 'Parquet_demo/*' PARQUET
file_access_error |
```

When computing the size of an external table, Vertica counts all data found in the location specified by the `COPY FROM` clause. If you have a directory that contains ORC and delimited files, for example, and you define your external table with `"COPY FROM *"` instead of `"COPY FROM *.orc"`, this table includes the size of the delimited files. (You would probably also

encounter errors when querying that external table.) When you query this table Vertica does not validate your table definition; it just uses the path to find files to report.

Calculating the Database Size

You can use your Vertica software until your columnar data reaches the maximum raw data size that the license agreement provides. This section describes when data is monitored, what data is included in the estimate, and the general methodology used to produce an estimate. For more information about monitoring for data size, see [Monitoring Database Size for License Compliance](#).

How Vertica Estimates Raw Data Size

Vertica uses statistical sampling to calculate an accurate estimate of the raw data size of the database. In this context, *raw data* means the uncompressed data stored in a single Vertica database. For the purpose of license size audit and enforcement, Vertica evaluates the raw data size as if the data had been exported from the database in text format, rather than as compressed data.

Vertica conducts your database size audit using statistical sampling. This method allows Vertica to estimate the size of the database without significantly affecting database performance. The trade-off between accuracy and impact on performance is a small margin of error, inherent in statistical sampling. Reports on your database size include the margin of error, so you can assess the accuracy of the estimate. To learn more about simple random sampling, see [Simple Random Sampling](#).

Excluding Data From Raw Data Size Estimate

Not all data in the Vertica database is evaluated as part of the raw data size. Specifically, Vertica excludes the following data:

- Multiple projections (underlying physical copies) of data from a logical database entity (table). Data appearing in multiple projections of the same table is counted only once.
- Data stored in temporary tables.
- Data accessible through external table definitions.

- Data that has been deleted, but that remains in the database. To understand more about deleting and purging data, see [Purging Deleted Data](#).
- Data stored in the WOS.
- Data stored in system and work tables such as monitoring tables, Data Collector tables, and Database Designer tables.
- Delimiter characters.
- Data stored in SET USING denormalized columns. For information on denormalized (flattened) tables, see [Flattened Tables](#).

Evaluating Data Type Footprint Size

Vertica treats the data sampled for the estimate as if it had been exported from the database in text format (such as printed from vsql). This means that Vertica evaluates the data type footprint sizes as follows:

- Strings and binary types (CHAR, VARCHAR, BINARY, VARBINARY) are counted as their actual size in bytes using UTF-8 encoding.
- Numeric data types are counted as if they had been printed. Each digit counts as a byte, as does any decimal point, sign, or scientific notation. For example, -123.456 counts as eight bytes (six digits plus the decimal point and minus sign).
- Date/time data types are counted as if they had been converted to text, including any hyphens or other separators. For example, a timestamp column containing the value for noon on July 4th, 2011 would be 19 bytes. As text, vsql would print the value as 2011-07-04 12:00:00, which is 19 characters, including the space between the date and the time.

Using AUDIT to Estimate Database Size

To supply a more accurate database size estimate than statistical sampling can provide, use the [AUDIT](#) function to perform a full audit. This function has parameters to set both the `error_tolerance` and `confidence_level`. Using one or both of these parameters increases or decreases the function's performance impact.

For instance, lowering the `error_tolerance` to zero (0) and raising the `confidence_level` to 100, provides the most accurate size estimate, and increases the performance impact of calling the AUDIT function. During a detailed, low error-tolerant audit, all of the data in the

database is dumped to a raw format to calculate its size. Since performing a stringent audit can significantly affect database performance, never perform a full audit of a production database. See [AUDIT](#) for details.

Note: Unlike estimating raw data size using statistical sampling, a full audit performs SQL queries on the full database contents, *including* the contents of the WOS.

Monitoring Database Size for License Compliance

Your Vertica license can include a data storage allowance. The allowance can consist of data in columnar tables, flex tables, or both types of data. The `AUDIT()` function estimates the columnar table data size and any flex table materialized columns. The `AUDIT_FLEX()` function estimates the amount of `__raw__` column data in flex or columnar tables. In regards to license data limits, data in `__raw__` columns is calculated at 1/10th the size of structured data. Monitoring data sizes for columnar and flex tables lets you plan either to schedule deleting old data to keep your database in compliance with your license, or to consider a license upgrade for additional data storage.

Note: An audit of columnar data includes flex table real and materialized columns, but not `__raw__` column data.

Viewing Your License Compliance Status

Vertica periodically runs an audit of the columnar data size to verify that your database is compliant with your license terms. You can view the results of the most recent audit by calling the `GET_COMPLIANCE_STATUS` function.

```
=> select GET_COMPLIANCE_STATUS();
          GET_COMPLIANCE_STATUS
-----
Raw Data Size: 2.00GB +/- 0.003GB
License Size : 4.000GB
Utilization  : 50%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

Periodically running `GET_COMPLIANCE_STATUS` to monitor your database's license status is usually enough to ensure that your database remains compliant with your license. If your database begins to near its columnar data allowance, you can use the other auditing functions described below to determine where your database is growing and how recent deletes affect the database size.

Manually Auditing Columnar Data Usage

You can manually check license compliance for all columnar data in your database using the `AUDIT_LICENSE_SIZE` function. This function performs the same audit that Vertica periodically performs automatically. The `AUDIT_LICENSE_SIZE` check runs in the background, so the function returns immediately. You can then query the results using `GET_COMPLIANCE_STATUS`.

Note: When you audit columnar data, the results include any flex table real and materialized columns, but not data in the `__raw__` column. Materialized columns are virtual columns that you have promoted to real columns. Columns that you define when creating a flex table, or which you add with `ALTER TABLE . . . ADD COLUMN` statements are real columns. All `__raw__` columns are real columns. However, since they consist of unstructured or semi-structured data, they are audited separately.

An alternative to `AUDIT_LICENSE_SIZE` is to use the `AUDIT` function to audit the size of the columnar tables in your entire database by passing an empty string to the function. This function operates synchronously, returning when it has estimated the size of the database.

```
=> SELECT AUDIT('');
      AUDIT
-----
      76376696
(1 row)
```

The size of the database is reported in bytes. The `AUDIT` function also allows you to control the accuracy of the estimated database size using additional parameters. See the entry for the `AUDIT` function in the SQL Reference Manual for full details. Vertica does not count the `AUDIT` function results as an official audit. It takes no license compliance actions based on the results.

Note: The results of the `AUDIT` function do not include flex table data in `__raw__` columns. Use the `AUDIT_FLEX` function to monitor data usage flex tables.

Manually Auditing __raw__ Column Data

You can use the [AUDIT_FLEX](#) function to manually audit data usage for flex or columnar tables with a __raw__ column. The function calculates the encoded, compressed data stored in ROS containers for any __raw__ columns. Materialized columns in flex tables are calculated by the [AUDIT](#) function. The [AUDIT_FLEX](#) results do not include data in the __raw__ columns of temporary flex tables.

Targeted Auditing

If audits determine that the columnar table estimates are unexpectedly large, consider schemas, tables, or partitions that are using the most storage. You can use the [AUDIT](#) function to perform targeted audits of schemas, tables, or partitions by supplying the name of the entity whose size you want to find. For example, to find the size of the `online_sales` schema in the [VMart](#) example database, run the following command:

```
=> SELECT AUDIT('online_sales');
      AUDIT
-----
35716504
(1 row)
```

You can also change the granularity of an audit to report the size of each object in a larger entity (for example, each table in a schema) by using the granularity argument of the [AUDIT](#) function. See the [AUDIT](#) function in the SQL Reference Manual.

Using Management Console to Monitor License Compliance

You can also get information about data storage of columnar data (for columnar tables and for materialized columns in flex tables) through the Management Console. This information is available in the database Overview page, which displays a grid view of the database's overall health.

- The needle in the license meter adjusts to reflect the amount used in megabytes.
- The grace period represents the term portion of the license.

- The Audit button returns the same information as the AUDIT() function in a graphical representation.
- The Details link within the License grid (next to the Audit button) provides historical information about license usage. This page also shows a progress meter of percent used toward your license limit.

Managing License Warnings and Limits

Term License Warnings and Expiration

The term portion of an Vertica license is easy to manage—you are licensed to use Vertica until a specific date. If the term of your license expires, Vertica alerts you with messages appearing in the Administration Tools and vsql. For example:

```
=> CREATE TABLE T (A INT);  
NOTICE: Vertica license is in its grace period  
HINT: Renew at http://www.vertica.com/  
CREATE TABLE
```

Contact Vertica at <http://www.vertica.com/about/contact-us/> as soon as possible to renew your license, and then [install the new license](#). After the grace period expires, Vertica stops processing queries.

Data Size License Warnings and Remedies

If your Vertica columnar license includes a raw data size allowance, Vertica periodically audits the size of your database to ensure it remains compliant with the license agreement. For details of this audit, see [Calculating the Database Size](#). You should also monitor your database size to know when it will approach licensed usage. Monitoring the database size helps you plan to either upgrade your license to allow for continued database growth or delete data from the database so you remain compliant with your license. See [Monitoring Database Size for License Compliance](#) for details.

If your database's size approaches your licensed usage allowance (above 75% of license limits), you will see warnings in the Administration Tools , vsql, and Management Console. You have two options to eliminate these warnings:

- Upgrade your license to a larger data size allowance.
- Delete data from your database to remain under your licensed raw data size allowance. The warnings disappear after Vertica's next audit of the database size shows that it is no longer close to or over the licensed amount. You can also manually run a database audit (see [Monitoring Database Size for License Compliance](#) for details).

If your database continues to grow after you receive warnings that its size is approaching your licensed size allowance, Vertica displays additional warnings in more parts of the system after a grace period passes.

If Your Vertica Premium Edition Database Size Exceeds Your Licensed Limits

If your Premium Edition database size exceeds your licensed data allowance, all successful queries from ODBC and JDBC clients return with a status of `SUCCESS_WITH_INFO` instead of the usual `SUCCESS`. The message sent with the results contains a warning about the database size. Your ODBC and JDBC clients should be prepared to handle these messages instead of assuming that successful requests always return `SUCCESS`.

Note: These warnings for Premium Edition are in addition to any warnings you see in Administration Tools, vsql, and Management Console.

If Your VerticaCommunity Edition Database Size Exceeds 1 Terabyte

If your Community Edition database size exceeds the limit of 1 terabyte, you will no longer be able to load or modify data in your database. In addition, you will not be able to delete data from your database.

To bring your database under compliance, you can choose to:

- Drop database tables. You can also consider truncating a table or dropping a partition. See [TRUNCATE TABLE](#) or [DROP_PARTITION](#).
- Upgrade to Vertica Premium Edition (or an evaluation license).

Exporting License Audit Results to CSV

You can use `admintools` to audit a database for license compliance and export the results in CSV format, as follows:

```
admintools -t license_audit [--password=password] --database=database] [--file=csv-file] [--quiet]
```

where:

- *database* must be a running database. If the database is password protected, you must also supply the password.
- `--file csv-file` directs output to the specified file. If *csv-file* already exists, the tool returns an error message. If this option is unspecified, output is directed to `stdout`.
- `--quiet` specifies that the tool should run in quiet mode; if unspecified, status messages are sent to `stdout`.

Running the `license_audit` tool is equivalent to invoking the following SQL statements:

```
select audit('');
select audit_flex('');
select * from dc_features_used;
select * from v_catalog.license_audits;
select * from v_catalog.user_audits;
```

Audit results include the following information:

- Log of used Vertica features
- Estimated database size
- Raw data size allowed by your Vertica license
- Percentage of licensed allowance that the database currently uses
- Audit timestamps

The following truncated example shows the raw CSV output that `license_audit` generates:

```
FEATURES_USED
features_used,feature,date,sum
features_used,metafunction::get_compliance_status,2014-08-04,1
features_used,metafunction::bootstrap_license,2014-08-04,1
...
```

LICENSE_AUDITS

```
license_audits, database_size_bytes, license_size_bytes, usage_percent, audit_start_timestamp, audit_end_
timestamp, confidence_level_percent, error_tolerance_percent, used_sampling, confidence_interval_lower_
bound_bytes, confidence_interval_upper_bound_bytes, sample_count, cell_count, license_name
license_audits, 808117909, 536870912000, 0.00150523690320551, 2014-08-04 23:59:00.024874-04, 2014-08-04
23:59:00.578419-04, 99, 5, t, 785472097, 830763721, 10000, 174754646, vertica
...
```

USER_AUDITS

```
user_audits, size_bytes, user_id, user_name, object_id, object_type, object_schema, object_name, audit_start_
timestamp, audit_end_timestamp, confidence_level_percent, error_tolerance_percent, used_
sampling, confidence_interval_lower_bound_bytes, confidence_interval_upper_bound_bytes, sample_
count, cell_count
user_audits, 812489249, 45035996273704962, dbadmin, 45035996273704974, DATABASE, , VMart, 2014-10-14
11:50:13.230669-04, 2014-10-14 11:50:14.069057-04, 99, 5, t, 789022736, 835955762, 10000, 174755178
```

AUDIT_SIZE_BYTES

```
audit_size_bytes, now, audit
audit_size_bytes, 2014-10-14 11:52:14.015231-04, 810584417
```

FLEX_SIZE_BYTES

```
flex_size_bytes, now, audit_flex
flex_size_bytes, 2014-10-14 11:52:15.117036-04, 11850
```

Configuring the Database

This section provides information about:

- The [Configuration Procedure](#)
- [Configuration Parameters](#)
- Designing a [logical schema](#)
- Creating the [physical schema](#)

You'll also want to set up a security scheme. See [Implementing Security](#).

See also implementing [locales](#) for international data sets.

Note: Before you begin this section, Micro Focus strongly recommends that you follow the [Tutorial](#) in Getting Started to quickly familiarize yourself with creating and configuring a fully-functioning example database.

Configuration Procedure

This section describes the tasks required to set up an Vertica database. It assumes that you have obtained a valid license key file, installed the Vertica rpm package, and run the installation script as described in [Installing Vertica](#).

You'll complete the configuration procedure using:

- Administration Tools

If you are unfamiliar with Dialog-based user interfaces, read [Using the Administration Tools Interface](#) before you begin. See also the [Administration Tools Reference](#) for details.

- vsql interactive interface
- Database Designer, described in [Creating a Database Design](#)

Note: You can also perform certain tasks using [Management Console](#). Those tasks point to the appropriate topic.

- Follow the configuration procedure in the order presented in this book.
- Micro Focus strongly recommends that you first use the [Tutorial](#) in Getting Started to experiment with creating and configuring a database.
- Although you may create more than one database (for example, one for production and one for testing), you may create only one active database for each installation of Vertica Analytics Platform
- The generic configuration procedure described here can be used several times during the development process and modified each time to fit changing goals. You can omit steps such as preparing actual data files and sample queries, and run the Database Designer without optimizing for queries. For example, you can create, load, and query a database several times for development and testing purposes, then one final time to create and load the production database.

Prepare Disk Storage Locations

You must create and specify directories in which to store your catalog and data files (physical schema). You can specify these locations when you install or configure the database, or later during database operations. Both the catalog and data directories must be owned by the database administrator.

The directory you specify for database catalog files (the catalog path) is used across all nodes in the cluster. For example, if you specify `/home/catalog` as the catalog directory, Vertica uses that catalog path on all nodes. The catalog directory should always be separate from any data file directories.

Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

The data path you designate is also used across all nodes in the cluster. Specifying that data should be stored in `/home/data`, Vertica uses this path on all database nodes.

Do not use a single directory to contain both catalog and data files. You can store the catalog and data directories on different drives, which can be either on drives local to the host (recommended for the catalog directory) or on a shared storage location, such as an external disk enclosure or a SAN.

Before you specify a catalog or data path, be sure the parent directory exists on all nodes of your database. Creating a database in `admintools` also creates the catalog and data directories, but the parent directory must exist on each node.

You do not need to specify a disk storage location during installation. However, you can do so by using the `--data-dir` parameter to the `install_vertica` script. See [Specifying Disk Storage Location During Installation](#)

See Also

- [Specifying Disk Storage Location on MC](#)
- [Specifying Disk Storage Location During Database Creation](#)
- [Configuring Disk Usage to Optimize Performance](#)
- [Using Shared Storage With Vertica](#)

Specifying Disk Storage Location During Installation

You can specify the disk storage location when you:

- Install Vertica (see below).
- [Create a database using the Administration Tools](#).
- [Install and configure Management Console](#).

Specifying Disk Storage Location When You Install

When you install Vertica, the `--data-dir` parameter in the `install_vertica` [script](#) lets you specify a directory to contain database data and catalog files. The script defaults to the database administrator's default home directory `/home/dbadmin`.

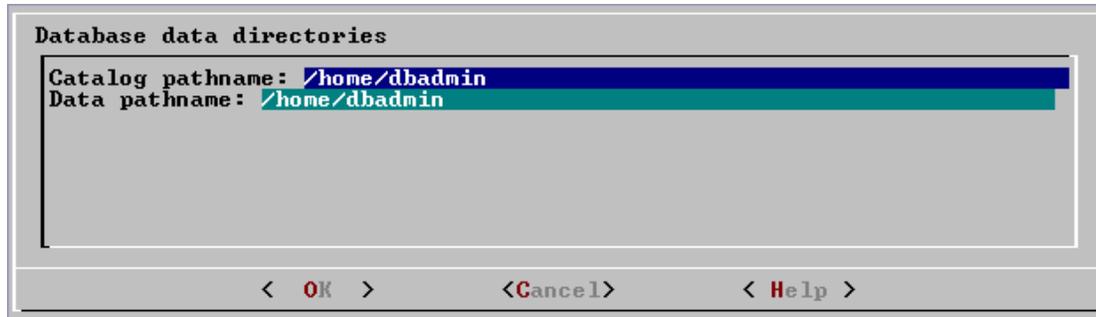
Important: Replace this default with a directory that has adequate space to hold your data and catalog files.

Requirements

- The data and catalog directory must exist on each node in the cluster.
- The directory on each node must be owned by the database administrator
- Catalog and data path names must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions will result in database creation failure.
- Vertica refuses to overwrite a directory if it appears to be in use by another database. Therefore, if you created a database for evaluation purposes, dropped the database, and want to reuse the database name, make sure that the disk storage location previously used has been completely cleaned up. See [Managing Storage Locations](#) for details.

Specifying Disk Storage Location During Database Creation

When you invoke the [Create Database](#) command in the Administration Tools, a dialog box allows you to specify the catalog and data locations. These locations must exist on each host in the cluster and must be owned by the database administrator.



When you click **OK**, Vertica automatically creates the following subdirectories:

```
catalog-pathname/database-name/node-name_catalog/data-pathname/database-name/node-name_data/
```

For example, if you use the default value (the database administrator's home directory) of `/home/dbadmin` for the Stock Exchange example database, the catalog and data directories are created on each node in the cluster as follows:

```
/home/dbadmin/Stock_Schema/stock_schema_node1_host01_catalog/home/dbadmin/Stock_Schema/stock_schema_node1_host01_data
```

Notes

- Catalog and data path names must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions will result in database creation failure.
- Vertica refuses to overwrite a directory if it appears to be in use by another database. Therefore, if you created a database for evaluation purposes, dropped the database, and want to reuse the database name, make sure that the disk storage location previously used has been completely cleaned up. See [Managing Storage Locations](#) for details.

Specifying Disk Storage Location on MC

You can use the MC interface to specify where you want to store database metadata on the cluster in the following ways:

- When you configure MC the first time
- When you create new databases using on MC

See [Configuring Management Console](#).

Configuring Disk Usage to Optimize Performance

Once you have created your initial storage location, you can add additional storage locations to the database later. Not only does this provide additional space, it lets you control disk usage and increase I/O performance by isolating files that have different I/O or access patterns. For example, consider:

- Isolating execution engine temporary files from data files by creating a separate storage location for temp space.
- Creating labeled storage locations and storage policies, in which selected database objects are stored on different storage locations based on measured performance statistics or predicted access patterns.

See [Managing Storage Locations](#) for details.

Using Shared Storage With Vertica

If using shared SAN storage, ensure there is no contention among the nodes for disk space or bandwidth.

- Each host must have its own catalog and data locations. Hosts cannot share catalog or data locations.
- Configure the storage so that there is enough I/O bandwidth for each node to access the storage independently.

Viewing Database Storage Information

You can view node-specific information on your Vertica cluster through the Management Console. See [Monitoring Using MC](#) for details.

Disk Space Requirements for Vertica

In addition to actual data stored in the database, Vertica requires disk space for several data reorganization operations, such as mergeout and [managing nodes](#) in the cluster. For best results, Micro Focus recommends that disk utilization per node be no more than sixty percent (60%) for a K-Safe=1 database to allow such operations to proceed.

In addition, disk space is temporarily required by certain query execution operators, such as hash joins and sorts, in the case when they cannot be completed in memory (RAM). Such operators might be encountered during queries, recovery, refreshing projections, and so on. The amount of disk space needed (known as temp space) depends on the nature of the queries, amount of data on the node and number of concurrent users on the system. By default, any unused disk space on the data disk can be used as temp space. However, Micro Focus recommends provisioning temp space separate from data disk space. See [Configuring Disk Usage to Optimize Performance](#).

Disk Space Requirements for Management Console

You can install Management Console on any node in the cluster, so it has no special disk requirements, other than disk space you allocate for your database cluster. See [Disk Space Requirements for Vertica](#).

Prepare the Logical Schema Script

Designing a logical schema for an Vertica database is no different from designing one for any other SQL database. Details are described more fully in [Designing a Logical Schema](#).

To create your logical schema, prepare a SQL script (plain text file, typically with an extension of .sql) that:

1. Creates additional schemas (as necessary). See [Using Multiple Schemas](#).
2. Creates the tables and column constraints in your database using the [CREATE TABLE](#) command.
3. Defines the necessary table constraints using the [ALTER TABLE](#) command.
4. Defines any views on the table using the [CREATE VIEW](#) command.

You can generate a script file using:

- A schema designer application.
- A schema extracted from an existing database.
- A text editor.
- One of the example database `example-name_define_schema.sql` scripts as a template. (See the example database directories in `/opt/vertica/examples`.)

In your script file, make sure that:

- Each statement ends with a semicolon.
- You use [data types](#) supported by Vertica, as described in the SQL Reference Manual.

Once you have created a database, you can test your schema script by executing it as described in [Create the Logical Schema](#). If you encounter errors, drop all tables, correct the errors, and run the script again.

Prepare Data Files

Prepare two sets of data files:

- Test data files. Use test files to test the database after the partial data load. If possible, use part of the actual data files to prepare the test data files.
- Actual data files. Once the database has been tested and optimized, use your data files for your initial [Bulk-Loading Data](#).

How to Name Data Files

Name each data file to match the corresponding table in the logical schema. Case does not matter.

Use the extension `.tbl` or whatever you prefer. For example, if a table is named `Stock_Dimension`, name the corresponding data file `stock_dimension.tbl`. When using multiple data files, append `_nnn` (where `nnn` is a positive integer in the range 001 to 999) to the file name. For example, `stock_dimension.tbl_001`, `stock_dimension.tbl_002`, and so on.

Prepare Load Scripts

Note: You can postpone this step if your goal is to test a logical schema design for validity.

Prepare SQL scripts to load data directly into physical storage using the [COPY...DIRECT](#) statement using `vsq`, or through ODBC as described in [Connecting to Vertica](#).

You need scripts that load the:

- Large tables
- Small tables

Micro Focus recommends that you load large tables using multiple files. To test the load process, use files of 10GB to 50GB in size. This size provides several advantages:

- You can use one of the data files as a sample data file for the Database Designer.
- You can load just enough data to [Perform a Partial Data Load](#) before you load the remainder.
- If a single load fails and rolls back, you do not lose an excessive amount of time.
- Once the load process is tested, for multi-terabyte tables, break up the full load in file sizes of 250–500GB.

See [Bulk-Loading Data](#) and the following additional topics for details:

- [Bulk-Loading Data](#)
- [Using Load Scripts](#)
- [Using Parallel Load Streams](#)
- [Enforcing Constraints](#)
- [About Load Errors](#)

Tip: You can use the load scripts included in the example databases in Getting Started as templates.

Create an Optional Sample Query Script

The purpose of a sample query script is to test your schema and load scripts for errors.

Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

Create an Empty Database

Two options are available for creating an empty database:

- Using the Management Console
- Using Administration Tools

Although you can create more than one database (for example, one for production and one for testing), there can be only one active database for each installation of Vertica Analytics Platform.

Creating a Database Name and Password

Database Names

Database names must conform to the following rules:

- Be between 1-30 characters
- Begin with a letter
- Follow with any combination of letters (upper and lowercase), numbers, and/or underscores.

Database names are case sensitive; however, Micro Focus strongly recommends that you do not create databases with names that differ only in case. For example, do not create a database called `mydatabase` and another called `MyDataBase`.

Database Passwords

Database passwords can contain letters, digits, and special characters listed in the next table. Passwords cannot include non-ASCII Unicode characters.

The allowed password length is between 0-100 characters. The database superuser can change a Vertica user's maximum password length using [ALTER PROFILE](#).

You use [Profiles](#) to specify and control password definitions. For instance, a profile can define the maximum length, reuse time, and the minimum number or required digits for a password, as well as other details.

The following table lists special (ASCII) characters that Vertica permits in database passwords. Special characters can appear anywhere in a password string. For example, `mypas$word` or `$mypasswordare` all valid.

Caution: Using special characters outside of the ones listed below could cause database instability.

Character	Description
#	pound sign
!	exclamation point
+	plus sign
*	asterisk
?	question mark
,	comma
.	period
/	forward slash
=	equals sign
~	tilde
-	minus sign
\$	dollar sign
_	underscore
:	colon
	space
"	double quote
'	single quote
%	percent sign
&	ampersand

(parenthesis
)	parenthesis
;	semicolon
<	less than sign
>	greater than sign
@	at sign
`	back quote
[square bracket
]	square bracket
\	backslash
^	caret
	vertical bar
{	curly bracket
}	curly bracket

See Also

- [Password Guidelines](#)
- [ALTER PROFILE](#)
- [CREATE PROFILE](#)
- [DROP PROFILE](#)

Create an Empty Database Using MC

You can create a new database on an existing Vertica cluster through the Management Console interface.

Database creation can be a long-running process, lasting from minutes to hours, depending on the size of the target database. You can close the web browser during the process and sign

back in to MC later; the creation process continues unless an unexpected error occurs. See the **Notes** section below the procedure on this page.

You currently need to use command line scripts to define the database schema and load data. Refer to the topics in [Configuration Procedure](#). You should also run the Database Designer, which you access through the Administration Tools, to create either a comprehensive or incremental design. Consider using the [Tutorial](#) in Getting Started to create a sample database you can start monitoring immediately.

How to Create an Empty Database on an MC-managed Cluster

1. If you are already on the **Databases and Clusters** page, skip to the next step; otherwise:
 - a. [Connect](#) to MC and sign in as an MC administrator.
 - b. On the [Home page](#), click **Existing Infrastructure** to view the Databases and Clusters page.
2. If no databases exist on the cluster, continue to the next step; otherwise:
 - a. If a database is running on the cluster on which you want to add a new database, select the database and click **Stop**.
 - b. Wait for the running database to have a status of *Stopped*.
3. Click the cluster on which you want to create the new database and click **Create Database**.
4. The Create Database wizard opens. Provide the following information:
 - Database name and password. See [Creating a Database Name and Password](#) for rules.
 - Optionally click **Advanced** to open the advanced settings and change the port, catalog path, and data path. By default the MC application/web server port is 5450 and paths are `/home/dbadmin`, or whatever you defined for the paths when you ran the Cluster Creation Wizard or the `install_vertica` script. Do not use the default agent port 5444 as a new setting for the MC port. See **MC Settings > Configuration** for port values.
5. Click **Continue**.
6. Select nodes to include in the database.

The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node

and clear **Include host in new database** to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the **Include** check box.

7. Click **Create** in the **Database Configuration** window to create the database on the nodes.

The creation process takes a few moments, after which the database starts and a **Success** message appears on the interface.

8. Click **OK** to close the success message.

The Manage page opens and displays the database nodes. Nodes not included in the database are colored gray, which means they are standby nodes you can include later. To add nodes to or remove nodes from your Vertica cluster, which are not shown in standby mode, you must run the `install_vertica` script.

Notes

- If warnings occur during database creation, nodes will be marked on the UI with an Alert icon and a message.
 - Warnings do not prevent the database from being created, but you should address warnings after the database creation process completes by viewing the database **Message Center** from the MC Home page.
 - Failure messages display on the database **Manage** page with a link to more detailed information and a hint with an actionable task that you must complete before you can continue. Problem nodes are colored red for quick identification.
 - To view more detailed information about a node in the cluster, double-click the node from the Manage page, which opens the **Node Details** page.
- To create MC users and grant them access to an MC-managed database, see [About MC Users](#) and [Creating an MC User](#).

See Also

- [Creating a Cluster Using MC](#)
- [Troubleshooting with MC Diagnostics](#)
- [Restarting MC](#)

Create a Database Using Administration Tools

1. Run the Administration Tools from your Administration Host as follows:

```
$ /opt/vertica/bin/admintools
```

If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, see [Notes for Remote Terminal Users](#).

2. Accept the license agreement and specify the location of your license file. For more information see [Managing Licenses](#) for more information.

This step is necessary only if it is the first time you have run the Administration Tools

3. On the Main Menu, click **Configuration Menu**, and click **OK**.
4. On the Configuration Menu, click **Create Database**, and click **OK**.
5. Enter the name of the database and an optional comment, and click **OK**. See [Creating a Database Name and Password](#) for naming guidelines and restrictions.
6. Establish the superuser password for your database.
 - To provide a password enter the password and click **OK**. Confirm the password by entering it again, and then click **OK**.
 - If you don't want to provide the password, leave it blank and click **OK**. If you don't set a password, Vertica prompts you to verify that you truly do not want to establish a superuser password for this database. Click **Yes** to create the database without a password or **No** to establish the password.

Caution: If you do not enter a password at this point, the superuser password is set to empty. Unless the database is for evaluation or academic purposes, Micro Focus strongly recommends that you enter a superuser password. See [Creating a Database Name and Password](#) for guidelines.

7. Select the hosts to include in the database from the list of hosts specified when Vertica was installed (`install_vertica -s`), and click **OK**.
8. Specify the directories in which to store the data and catalog files, and click **OK**.

Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

9. Catalog and data path names must contain only alphanumeric characters and cannot have leading spaces. Failure to comply with these restrictions results in database creation failure.

For example:

Catalog pathname: /home/dbadmin

Data Pathname: /home/dbadmin

10. Review the **Current Database Definition** screen to verify that it represents the database you want to create, and then click **Yes** to proceed or **No** to modify the database definition.
11. If you click **Yes**, Vertica creates the database you defined and then displays a message to indicate that the database was successfully created.

Note: : For databases created with 3 or more nodes, Vertica automatically sets K-safety to 1 to ensure that the database is fault tolerant in case a node fails. For more information, see [Failure Recovery](#) in the Administrator's Guide and [MARK_DESIGN_KSAFE](#)

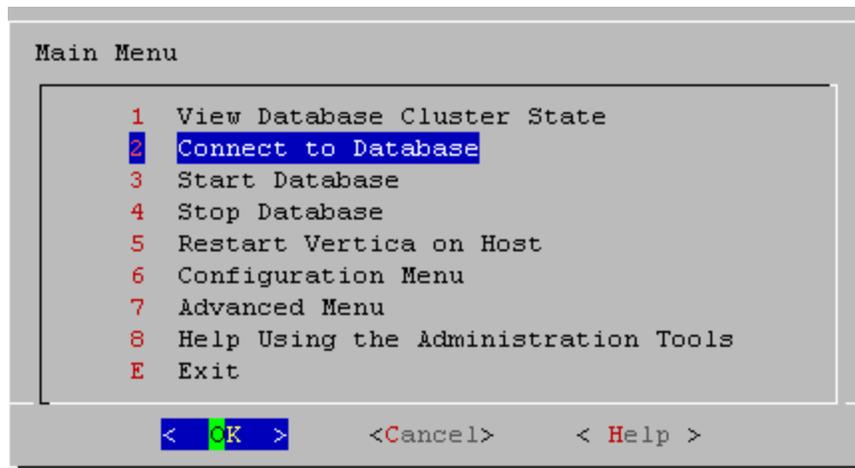
12. Click **OK** to acknowledge the message.

Create the Logical Schema

1. **Connect to the database.**

In the Administration Tools Main Menu, click **Connect to Database** and click **OK**.

See [Connecting to the Database](#) for details.



The vsql welcome script appears:

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.
Type: \h or \? for help with vsql commands
      \g or terminate with semicolon to execute query
      \q to quit

=>
```

2. Run the logical schema script

Using the [\i meta-command](#) in vsql to run the SQL [logical schema script](#) that you prepared earlier.

3. Disconnect from the database

Use the [\q meta-command](#) in vsql to return to the Administration Tools.

Perform a Partial Data Load

Micro Focus recommends that for large tables, you perform a partial data load and then test your database before completing a full data load. This load should load a representative amount of data.

1. Load the small tables.

Load the small table data files using the SQL [load scripts](#) and [data files](#) you prepared earlier.

2. Partially load the large tables.

Load 10GB to 50GB of table data for each table using the SQL [load scripts](#) and [data files](#) that you prepared earlier.

For more information about projections, see [Physical Schema](#) in Vertica Concepts.

Test the Database

Test the database to verify that it is running as expected.

Check queries for syntax errors and execution times.

1. Use the vsql [\timing meta-command](#) to enable the display of query execution time in milliseconds.
2. Execute the SQL sample query script that you prepared earlier.
3. Execute several ad hoc queries.

Optimize Query Performance

Optimizing the database consists of optimizing for compression and tuning for queries. (See [Creating a Database Design](#).)

To optimize the database, use the Database Designer to create and deploy a design for optimizing the database. See the [Tutorial](#) in Getting Started for an example of using the Database Designer to create a Comprehensive Design.

After you have run the Database Designer, use the techniques described in [Query Optimization](#) in Analyzing Data to improve the performance of certain types of queries.

Note: The database response time depends on factors such as type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if the response time is already short, e.g., less than 10 seconds, or the response time is not hardware bound.

Complete the Data Load

To complete the load:

1. Monitor system resource usage.

Continue to run the `top`, `free`, and `df` utilities and watch them while your load scripts are running (as described in [Monitoring Linux Resource Usage](#)). You can do this on any or all nodes in the cluster. Make sure that the system is not swapping excessively (watch `kswapd` in `top`) or running out of swap space (watch for a large amount of used swap space in `free`).

Note: Vertica requires a dedicated server. If your loader or other processes take up significant amounts of RAM, it can result in swapping.

2. Complete the large table loads.

Run the remainder of the large table load scripts.

Test the Optimized Database

Check query execution times to test your optimized design:

1. Use the `vsql \timing` meta-command to enable the display of query execution time in milliseconds.

Execute a SQL sample query script to test your schema and load scripts for errors.

Note: Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

2. Execute several ad hoc queries

- a. Run Administration Tools and select [Connect to Database](#).
- b. Use the `\i` meta-command to execute the query script; for example:

```
vmartdb=> \i vmart_query_03.sql customer_name | annual_income
-----+-----
James M. McNulty |          999979
Emily G. Vogel   |          999998
(2 rows)
Time: First fetch (2 rows): 58.411 ms. All rows formatted: 58.448 ms
vmartdb=> \i vmart_query_06.sql
store_key | order_number | date_ordered
-----+-----
```

```
45 | 202416 | 2004-01-04
113 | 66017 | 2004-01-04
121 | 251417 | 2004-01-04
24 | 250295 | 2004-01-04
9 | 188567 | 2004-01-04
166 | 36008 | 2004-01-04
27 | 150241 | 2004-01-04
148 | 182207 | 2004-01-04
198 | 75716 | 2004-01-04
(9 rows)
Time: First fetch (9 rows): 25.342 ms. All rows formatted: 25.383 ms
```

Once the database is optimized, it should run queries efficiently. If you discover queries that you want to optimize, you can modify and update the design. See [Incremental Design](#) in the Administrator's Guide.

Set Up Incremental (Trickle) Loads

Once you have a working database, you can use trickle loading to load new data while concurrent queries are running.

Trickle load is accomplished by using the COPY command (without the DIRECT keyword) to load 10,000 to 100,000 rows per transaction into the WOS. This allows Vertica to batch multiple loads when it writes data to disk. While the COPY command defaults to loading into the WOS, it will write ROS if the WOS is full.

See [Trickle Loading Data](#) for details.

See Also

- [COPY](#)
- [Loading Data Through ODBC](#)

Implement Locales for International Data Sets

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate `GROUP BY` and `ORDER BY` clauses, joins, and the analytic `ORDER BY` clause.

The default locale for a Vertica database is `en_US@collation=binary` (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default `en_US@collation=binary` collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to `en_US_POSIX`. You can set the locale back to the default locale and collation by issuing the `vsql` meta-command `\locale`. For example:

```
=> set locale to '';
INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation=binary;
```

You can set locale through [ODBC](#), [JDBC](#), and [ADO.net](#).

ICU Locale Support

Vertica uses the ICU library for locale support; you must specify locale using the ICU locale syntax. The locale used by the database session is not derived from the operating system (through the `LANG` variable), so Micro Focus recommends that you set the `LANG` for each node running `vsql`, as described in the next section.

While ICU library services can specify collation, currency, and calendar preferences, Vertica supports only the collation component. Any keywords not relating to collation are rejected. Projections are always collated using the `en_US@collation=binary` collation regardless of the session collation. Any locale-specific collation is applied at query time.

The `SET DATESTYLE TO . . .` command provides some aspects of the calendar, but Vertica supports only dollars as currency.

Changing DB Locale for a Session

This examples sets the session locale to Thai.

1. At the operating-system level for each node running `vsq`, set the `LANG` variable to the locale language as follows:

```
export LANG=th_TH.UTF-8
```

Note: If setting the `LANG=` as shown does not work, the operating system support for locales may not be installed.

2. For each Vertica session (from ODBC/JDBC or `vsq`) set the language locale.

From `vsq`:

```
\locale th_TH
```

3. From ODBC/JDBC:

```
"SET LOCALE TO th_TH;"
```

4. In PUTTY (or ssh terminal), change the settings as follows:

```
settings > window > translation > UTF-8
```

5. Click **Apply** and then click **Save**.

All data loaded must be in UTF-8 format, not an ISO format, as described in [Loading UTF-8 Format Data](#). Character sets like ISO 8859-1 (Latin1), which are incompatible with UTF-8, are not supported, so functions like `SUBSTRING` do not work correctly for multibyte characters. Thus, settings for locale should *not* work correctly. If the translation setting `ISO-8859-11:2001` (Latin/Thai) works, the data is loaded incorrectly. To convert data correctly, use a utility program such as Linux [iconv](#).

Note: The maximum length parameter for `VARCHAR` and `CHAR` data type refers to the number of octets (bytes) that can be stored in that field, not the number of characters. When using multi-byte UTF-8 characters, make sure to size fields to accommodate from 1

to 4 bytes per character, depending on the data.

See Also

- [Supported Locales](#)
- [About Locale](#)
- [SET LOCALE](#)
- [ICU User Guide](#)

Specify the Default Locale for the Database

After you start the database, the default locale configuration parameter, `DefaultSessionLocale`, sets the initial locale. You can override this value for individual sessions.

To set the locale for the database, use the configuration parameter as follows:

```
=> ALTER DATABASE mydb SET DefaultSessionLocale = 'ICU-Locale-identifier';
```

For example:

```
=> ALTER DATABASE mydb SET DefaultSessionLocale = 'en_GB';
```

Override the Default Locale for a Session

To override the default locale for a specific session, use one of the following commands:

- The `vsq` command

```
\locale <ICU-Locale-identifier>;
```

For example:

```
=> \locale en_GBINFO:  
INFO 2567: Canonical locale: 'en_GB'  
Standard collation: 'LEN'  
English (United Kingdom)
```

- The statement `SET LOCALE TO <ICU-locale-identifier>`.

```
=> SET LOCALE TO en_GB;  
INFO 2567: Canonical locale: 'en_GB'  
Standard collation: 'LEN'  
English (United Kingdom)
```

You can also use the [Specifying Locale: Short Form](#) of a locale in either of these commands:

```
=> SET LOCALE TO LEN;  
INFO 2567: Canonical locale: 'en'  
Standard collation: 'LEN'  
English
```

```
=> \locale LEN  
INFO 2567: Canonical locale: 'en'  
Standard collation: 'LEN'  
English
```

You can use these commands to override the locale as many times as needed during a database session. The session locale setting applies to any subsequent commands issued in the session.

See Also

- [SET LOCALE](#)

Server versus Client Locale Settings

Vertica differentiates database server locale settings from client application locale settings:

- Server locale settings only impact collation behavior for server-side query processing.
- Client applications verify that locale is set appropriately in order to display characters correctly.

The following sections describe best practices to ensure predictable results.

Server Locale

The server session locale should be set as described in [Specify the Default Locale for the Database](#). If locales vary across different sessions, set the server locale at the start of each session from your client.

vsqI Client

- If the database does not have a default session locale, [set the server locale for the session to the desired locale](#).
- The locale setting in the terminal emulator where the vsqI client runs should be set to be equivalent to session locale setting on the server side (ICU locale). By doing so, the data is collated correctly on the server and displayed correctly on the client.
- All input data for vsqI should be in UTF-8, and all output data is encoded in UTF-8
- Vertica does not support non UTF-8 encodings and associated locale values; .
- For instructions on setting locale and encoding, refer to your terminal emulator documentation.

ODBC Clients

- ODBC applications can be either in ANSI or Unicode mode. If the user application is Unicode, the encoding used by ODBC is UCS-2. If the user application is ANSI, the data must be in single-byte ASCII, which is compatible with UTF-8 used on the database server. The ODBC driver converts UCS-2 to UTF-8 when passing to the Vertica server and converts data sent by the Vertica server from UTF-8 to UCS-2.
- If the user application is not already in UCS-2, the application must convert the input data to UCS-2, or unexpected results could occur. For example:
 - For non-UCS-2 data passed to ODBC APIs, when it is interpreted as UCS-2, it could result in an invalid UCS-2 symbol being passed to the APIs, resulting in errors.
 - The symbol provided in the alternate encoding could be a valid UCS-2 symbol. If this occurs, incorrect data is inserted into the database.
- If the database does not have a default session locale, ODBC applications should set the desired server session locale using `SQLSetConnectAttr` (if different from database wide setting). By doing so, you get the expected collation and string functions behavior on the server.

JDBC and ADO.NET Clients

- JDBC and ADO.NET applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. The same cautions apply as for ODBC if this encoding is violated.
- The JDBC and ADO.NET drivers convert UTF-16 data to UTF-8 when passing to the Vertica server and convert data sent by Vertica server from UTF-8 to UTF-16.
- If there is no default session locale at the database level, JDBC and ADO.NET applications should set the correct server session locale by executing the [SET LOCALE TO](#) command in order to get the expected collation and string functions behavior on the server. For more information, see [SET LOCALE](#).

Change Transaction Isolation Levels

By default, Vertica uses the `READ COMMITTED` isolation level for all sessions. You can change the default isolation level for the database or for a given session.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the `SERIALIZABLE` isolation level to ensure consistency.

Database Isolation Level

The configuration parameter [TransactionIsolationLevel](#) specifies the database isolation level, and is used as the default for all sessions. Use [ALTER DATABASE](#) to change the default isolation level. For example:

```
=> ALTER DATABASE mydb SET TransactionIsolationLevel = 'SERIALIZABLE';  
ALTER DATABASE  
=> ALTER DATABASE mydb SET TransactionIsolationLevel = 'READ COMMITTED';  
ALTER DATABASE
```

Changes to the database isolation level only apply to future sessions. Existing sessions and their transactions continue to use their original isolation level.

Use [SHOW CURRENT](#) to view the database isolation level:

```
=> SHOW CURRENT TransactionIsolationLevel;  
level | name | setting
```

```
-----+-----+-----  
DATABASE | TransactionIsolationLevel | READ COMMITTED  
(1 row)
```

Session Isolation Level

[SET SESSION CHARACTERISTICS AS TRANSACTION](#) changes the isolation level for a specific session. For example:

```
=> SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET
```

Use [SHOW](#) to view the current session's isolation level:

```
=> SHOW TRANSACTION_ISOLATION;
```

See Also

[Transactions](#)

Configuration Parameters

Configuration parameters are settings that affect database behavior. You can use configuration parameters to enable, disable, or tune features related to different database aspects like Tuple Mover, security, Database Designer, or projections. Configuration parameters have default values, stored in the Vertica database.

You can modify certain parameters to configure your Vertica database in two ways:

- Management Console [browser-based interface](#)
- [VSQL statements](#)

Before you modify a database parameter, review all documentation about the parameter to determine the context under which you can change it. Some parameter changes require a database restart to take effect. The `CHANGE_REQUIRES_RESTART` column in the system table [CONFIGURATION_PARAMETERS](#) indicates whether a parameter requires a restart.

Managing Configuration Parameters: Management Console

To change database settings for any MC-managed database, click the **Settings** tab at the bottom of the Overview, Activity, or Manage pages. The database must be running.

The Settings page defaults to parameters in the General category. To change other parameters, click an option from the tab panel on the left.

		Current value	Default value
Tuple mover	Analyze row count interval	60 seconds	60
Epoch mgmt	Compress network data	false	false
Monitoring	Maximum client sessions	50	50
Password	Transaction isolation level	READ COMMITTED	READ COMMITTED
Profiling	Transaction mode	READ WRITE	READ WRITE
Snapshot			
I18N			
License			

Some settings require you to restart the database, and MC prompts you to do so. You can ignore the prompt, but those changes take effect only after the database restarts.

Some settings are specific to Management Console, such as changing MC or agent port assignments. For more information, see [Managing MC Settings](#) in Using Management Console.

Managing Configuration Parameters: VSQL

You can configure all parameters at database scope. Some parameters can also be set and cleared at node and session scopes.

Caution: Vertica is designed to operate with minimal configuration changes. Be careful to set and change configuration parameters according to documented guidelines.

For detailed information about managing configuration parameters, see:

- [Viewing Configuration Parameter Values](#)
- [Setting Configuration Parameter Values](#)
- [Clearing Configuration Parameters](#)

Viewing Configuration Parameter Values

You can view active configuration parameter values in two ways:

- [SHOW statements](#)
- [Query related system tables](#)

SHOW Statements

Use the following SHOW statements to view active configuration parameters:

- **SHOW CURRENT**: Returns settings of active configuration parameter values. Vertica checks settings at all levels, in the following ascending order of precedence:
 - session
 - node
 - database

If no values are set at any scope, SHOW CURRENT returns the parameter's default value.

- **SHOW DATABASE**: Displays configuration parameter values set for the database.
- **SHOW SESSION**: Displays configuration parameter values set for the current session.
- **SHOW NODE**: Displays configuration parameter values set for a node.

If a configuration parameter requires a restart to take effect, the values in a SHOW CURRENT statement might differ from values in other SHOW statements. To see which parameters require restart, query the [CONFIGURATION_PARAMETERS](#) system table.

System Tables

You can query two system tables for configuration parameters:

- [SESSION_PARAMETERS](#) returns session-scope parameters.
- [CONFIGURATION_PARAMETERS](#) returns parameters for all scopes: database, node, and session.

Setting Configuration Parameter Values

You can set configuration parameters at three scopes:

- Database
- Node
- Session

Database Scope

You can set one or more parameter values at the database scope with [ALTER DATABASE . .SET](#):

```
ALTER DATABASE dbname SET parameter-name = value[,...];
```

For example:

```
ALTER DATABASE mydb SET AnalyzeRowCountInterval = 3600, FailoverToStandbyAfter = '5 minutes';
```

Node Scope

You can set one or more parameter values at the node scope with [ALTER NODE . .SET](#):

```
ALTER NODE node-name SET parameter-name = value[,...];
```

For example, to prevent clients from connecting to `v_vmart_node0001`, set the `MaxClientSessions` configuration parameter to `0`:

```
=> ALTER NODE v_vmart_node0001 SET MaxClientSessions = 0;
```

Session Scope

You can set one or more parameter values at the session scope with [ALTER SESSION . .SET](#):

```
ALTER SESSION SET parameter-name = value[,...];
```

For example:

```
=> ALTER SESSION SET ForceUDxFencedMode = 1;
```

Clearing Configuration Parameters

You can clear configuration parameter settings at three scopes:

- Database
- Node
- Session

Database Scope

ALTER DATABASE . . CLEAR clears one or more parameter values at the database scope, and resets them to their default values as follows:

```
ALTER DATABASE dbname CLEAR parameter-name[, ...];
```

For example:

```
ALTER DATABASE mydb CLEAR AnalyzeRowCountInterval, FailoverToStandbyAfter;
```

Node Scope

ALTER NODE . . CLEAR clears one or more parameter values at the node scope, and resets them to their database settings, if any. If the parameters are not set at the database scope, Vertica resets them to their default value.

```
ALTER NODE node-name CLEAR parameter-name[, ...];
```

The following example clears MaxClientSessions on node v_vmart_node0001:

```
ALTER NODE v_vmart_node0001 CLEAR MaxClientSessions;
```

Session Scope

ALTER SESSION . . CLEAR clears one or more parameter values at the session scope, and resets them to their node or database settings, if any. If the parameters are not set at either scope, Vertica resets them to their default value.

```
ALTER SESSION CLEAR parameter-name[, ...];
```

For example:

```
=> ALTER SESSION CLEAR ForceUDxFencedMode;
```

Configuration Parameter Categories

Vertica configuration parameters are grouped into the following categories:

[General Parameters](#)

[Tuple Mover Parameters](#)

[Projection Parameters](#)

[Epoch Management Parameters](#)

[Monitoring Parameters](#)

[Profiling Parameters](#)

[Security Parameters](#)

[Database Designer Parameters](#)

[Internationalization Parameters](#)

[Data Collector Parameters](#)

[Text Search Parameters](#)

[Kerberos Authentication Parameters](#)

[Hadoop Parameters](#)

[User-Defined Session Parameters](#)

[Constraint Enforcement Parameters](#)

General Parameters

You use these general parameters to configure Vertica.

Parameter	Description
AnalyzeRowCountInterval	Specifies how often Vertica checks the number of projection rows and whether the threshold set by ARCCommitPercentage has been crossed. For more information, see Collecting Statistics .

Parameter	Description
	<p>Default Value: 60 seconds</p>
<p>ApplyEventsDuringSALCheck</p>	<p>When enabled, Vertica uses catalog events to filter out dropped corrupt partitions during node startup. Dropping corrupt partitions can speed node recovery.</p> <p>When disabled, Vertica reports corrupt partitions, but takes no action. Leaving corrupt partitions in place can reset the current projection checkpoint epoch to the epoch before the corruption occurred.</p> <p>This parameter has no effect on unpartitioned tables.</p> <p>Default Value: 0</p>
<p>ApportionedFileMinimumPortionsSizeKB</p>	<p>Specifies the minimum portion size (in kilobytes) for use with apportioned file loads. Vertica apports a file load across multiple nodes only if:</p> <ul style="list-style-type: none"> • The load can be divided into portions at least equaling this value. • EnableApportionedFileLoad and EnableApportionLoad are set to 1 (enabled). <p>See also EnableApportionLoad and EnableApportionedFileLoad.</p> <p>Default Value: 1024</p>
<p>ARCCommitPercentage</p>	<p>Sets the threshold percentage of WOS to ROS rows, which determines when to aggregate projection row counts and commit the result to the catalog. Vertica performs this action when the WOS to ROS percentage exceeds this setting.</p> <p>Default Value: 3 (percent)</p>

Parameter	Description
BlockedSocketGracePeriod	<p>Sets how long a session socket remains blocked while awaiting client input or output for a given query.</p> <p>Default Value: None (Socket blocking can continue indefinitely.)</p> <p>See Handling Session Socket Blocking.</p>
CatalogCheckpointPercent	<p>Specifies the threshold at which a checkpoint is created for the database catalog.</p> <p>By default, this parameter is set to 50 (percent), so when transaction logs reach 50% of the size of the last checkpoint, Vertica adds a checkpoint. Each checkpoint demarcates all changes to the catalog since the last checkpoint.</p> <p>Default Value: 50 (percent)</p>
ClusterSequenceCacheMode	<p>Indicates whether the initiator node requests cache for other nodes in a cluster, and then sends cache to other nodes along with the execution plan. Enabled by default. When disabled, all nodes request their own cache.</p> <p>Default Value: 1 (enabled)</p> <p>Valid Values:</p> <ul style="list-style-type: none">• 1: Initiator node requests cache.• 0: All nodes request their own cache. <p>See Distributing Named Sequences.</p>
CompressCatalogOnDisk	<p>Compresses the size of the catalog on disk when enabled (value set to 1 or 2).</p> <p>Default Value: 1</p> <p>Valid Values:</p>

Parameter	Description
	<ul style="list-style-type: none"> • 0: No compression. • 1: Compress checkpoints and system transaction logs. • 2: Compress checkpoints and all transaction logs. <p>Consider enabling this parameter if the catalog disk partition is small (<50 GB) and the metadata is large (hundreds of tables, partitions, or nodes).</p>
CompressNetworkData	<p>Compresses all data sent over the internal network when enabled (value set to 1). This compression speeds up network traffic at the expense of added CPU load. If the network is throttling database performance, enable compression to correct the issue.</p> <p>Default Value: 0</p>
CopyFromVerticaWithIdentity	<p>Allows COPY FROM VERTICA and EXPORT TO VERTICA to load values into Identity (or Auto-increment) columns. The destination Identity column is not incremented automatically. To disable the default behavior, set this parameter to 0 (zero).</p> <p>Default Value: 1</p>
DatabaseHeartBeatInterval	<p>Determines the interval (in seconds) at which each node performs a health check and communicates a heartbeat. If a node does not receive a message within five times of the specified interval, the node is evicted from the cluster. Setting the interval to 0 disables the feature.</p> <p>Default Value: 120</p>

Parameter	Description
	See Automatic Eviction of Unhealthy Nodes .
DivideZeroByZeroThrowsError	<p>If set to 1, returns an error if a division by zero operation is requested. Otherwise, returns 0 as the result of a division by zero operation.</p> <p>Default Value: 1</p>
EnableApportionedChunkingInDefaultLoadParser	<p>Enables the built-in parser for delimited files to take advantage of both apportioned load and cooperative parse for potentially better performance.</p> <p>Default Value: 1</p>
EnableApportionedFileLoad	<p>Enables automatic apportioning across nodes of file loads using COPY FROM. Vertica attempts to apportion the load if:</p> <ul style="list-style-type: none"> • This parameter and EnableApportionLoad are both enabled. • The parser supports apportioning. • The load is divisible into portion sizes of at least the value of ApportionedFileMinimumPortionSizeKB. <p>Setting this parameter does not guarantee that loads will be apportioned, but disabling it guarantees that they will not be.</p> <p>Default Value: 1</p> <p>See Using Parallel Load Streams.</p>
EnableApportionLoad	Enables automatic apportioning across

Parameter	Description
	<p>nodes of data loads using COPY WITH SOURCE. Vertica attempts to apportion the load if:</p> <ul style="list-style-type: none"> • This parameter is enabled. • The source and parser both support apportioning. <p>Setting this parameter does not guarantee that loads will be apportioned, but disabling it guarantees that they will not be.</p> <p>Default Value: 1</p> <p>See Using Parallel Load Streams.</p>
<p>EnableBetterFlexTypeGuessing</p>	<p>Enables more accurate type guessing when assigning data types to non-string keys in a flex table <code>__raw__</code> column with <code>COMPUTE_FLEXTABLE_KEYS</code> or <code>COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW</code>. This option is on by default. Turning this option off uses a limited set of Vertica data type assignments.</p> <p>Default Value: 1</p> <p>See Setting Flex Table Configuration Parameters.</p>
<p>EnableCooperativeParse</p>	<p>Implements multi-threaded parsing capabilities on a node. You can use this parameter for both delimited and fixed-width loads. Enabled by default.</p> <p>Default Value: 1</p>
<p>EnableDataTargetParallelism</p>	<p>Enables multiple threads for sorting and writing data to ROS, improving data loading performance. Enabled by default.</p> <p>Default Value: 1</p>

Parameter	Description
EnableForceOuter	<p>Determines whether Vertica uses a table's <code>force_outer</code> value to implement a join. For more information, see Controlling Join Inputs.</p> <p>Default Value: 0 (forced join inputs disabled)</p>
EnableMetadataMemoryTracking	<p>Enables Vertica to track memory used by database metadata in the METADATA resource pool.</p> <div data-bbox="857 714 1404 1365" style="background-color: #f0f0f0; padding: 10px;"> <p>Note: You may notice a performance degradation when loading data in 8.0 SP1 and later for some COPY statements, typically for loads with a large number of columns and large catalogs, in comparison with Vertica 8.0. This degradation occurs because the number of threads used to load the data is computed based on the available memory in the General Pool. Tracking the catalog size separately in the new METADATA resource pool can reduce amount of memory available in the General Pool and can affect this thread calculation.</p> </div> <p>Default Value: 1</p>
EnableResourcePoolCPUAffinity	<p>Aligns queries to the resource pool of the processing CPU. When disabled (value is set to 0), queries run on any CPU, regardless of the <code>CPU_AFFINITY_SET</code> of the resource pool. Enabled by default.</p> <p>Default Value: 1</p>
EnableStorageBundling	<p>Enables storing multiple ROS containers as a single file. Each ROS must be less than the size specified in</p>

Parameter	Description
	<p>MaxBundleableROSSizeKB. In environments with many small storage files, bundling improves the performance of any file-intensive operations, including backups, restores, mergeouts and moveouts.</p> <p>Default Value: 1</p>
<p>EnableUniquenessOptimization</p>	<p>Enables query optimization that is based on guaranteed uniqueness of column values. Columns that can be guaranteed to include unique values include:</p> <ul style="list-style-type: none"> • Columns that are defined with AUTO_INCREMENT or IDENTITY constraints • Primary key columns where key constraints are enforced • Columns that are constrained to unique values, either individually or as a set <p>Default Value: 1 (enabled)</p>
<p>EnableWithClauseMaterialization</p>	<p>Enables materialization of WITH clause results. When materialization is enabled, Vertica evaluates each WITH clause once and stores results in a temporary table. This parameter can only be set at session level.</p> <p>Default Value: 0 (disabled)</p> <p>See WITH Clauses in SELECT in Analyzing Data.</p>
<p>ExternalTablesExceptionsLimit</p>	<p>Determines the maximum number of COPY exceptions and rejections allowed when a SELECT statement references an external table. Set to -1 to remove any exceptions limit. See Validating External</p>

Parameter	Description
	<p>Tables.</p> <p>Default Value: 100</p>
FailoverToStandbyAfter	<p>Specifies the length of time that an active standby node waits before taking the place of a failed node.</p> <p>This parameter is set to an interval literal.</p> <p>Default Value: None</p>
FencedUDxMemoryLimitMB	<p>Sets the maximum amount of memory, in megabytes (MB), that a fenced-mode UDF can use. If a UDF attempts to allocate more memory than this limit, that attempt triggers an exception. For more information, see Fenced Mode in Extending Vertica.</p> <p>Default Value: -1 (no limit)</p>
FlexTableDataTypeGuessMultiplier	<p>Specifies a multiplier that the COMPUTE_FLEXTABLE_KEYS and COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW functions use when assigning a data type and column width for the flex keys table. Both functions assign each key a data type, and multiply the longest key value by this factor to estimate column width.</p> <p>Default Value: 2.0: The column width multiplier. Must be a value in the following range:</p> <p>Any value that results in a column width neither less than 20 bytes nor greater than FlexTableRawSize. This range is a cap to round sizes up or down, accordingly.</p> <p>See Setting Flex Table Configuration Parameters.</p>

Parameter	Description
FlexTableRowSize	<p>Specifies the default column width for the <code>__raw__</code> column of new flex tables.</p> <p>Default Value: 130000</p> <p>Value range: 1 – 32000000</p>
JavaBinaryForUDx	<p>Sets the full path to the Java executable that Vertica uses to run Java UDxs. See Installing Java on Vertica Hosts in <i>Extending Vertica</i>.</p>
JavaClassPathForUDx	<p>Sets the Java classpath for the JVM that executes Java UDxs. This parameter must list all directories containing JAR files that Java UDxs import.</p> <p>Default Value:</p> <p><code>\${vertica_home}/packages/hcat/lib/*</code></p> <p>See Handling Java UDx Dependencies in <i>Extending Vertica</i>.</p>
MaxAutoSegColumns	<p>Specifies the number of columns (0–1024) to segment automatically when creating auto-projections from COPY and INSERT INTO statements. Setting this parameter to zero (0) uses all columns in the hash segmentation expression.</p> <p>Default Value: 32</p>
MaxBundleableROSSizeKB	<p>Specifies the minimum size, in kilobytes, of an independent ROS file. When <code>EnableStorageBundling</code> is true, Vertica bundles storage container ROS files below this size into a single file. Bundling improves the performance of any file-intensive operations, including backups, restores, mergeouts and moveouts.</p> <p>If you enable storage bundling and specify this parameter with a value of 0, Vertica bundles <code>.fdb</code> and <code>.pidx</code> files without</p>

Parameter	Description
	<p>bundling other storage container files.</p> <p>Default Value: 1024</p>
MaxClientSessions	<p>Determines the maximum number of client sessions that can run on a single node of the database. The default value allows for five additional administrative logins. These logins prevent DBAs from being locked out of the system if non-dbadmin users reach the login limit.</p> <div data-bbox="857 716 1403 1031" style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Tip: Setting this parameter to 0 prevents new client sessions from being opened while you are shutting down the database. Restore the parameter to its original setting after you restart the database. For details, see Managing Sessions.</p> </div> <p>Default Value: 50 user logins and 5 additional administrative logins</p>
MaxTieredPoolScale	<p>Specifies the threshold for allocating chunks of memory from the system or dedicated pools for Vertica system components—for example, for WOS containers. Above this threshold, Vertica allocates memory from the system (malloc), and returns it when no longer needed. Below this threshold, Vertica allocates memory from dedicated pools (talloc), which manage smaller memory allocations more efficiently.</p> <p>Tiered memory management facilitates more efficient use of resources, and minimizes the risk of any one process (such as the database catalog) from preemptively allocating and retaining excessive amounts of memory.</p>

Parameter	Description
	<p>Set this parameter to the desired power of 2, any value ≥ 20. For example, for 128 MB, set the parameter to 27 (2^{27}).</p> <p>This parameter cannot be set below 20 (1MB). Any allocation that is less than 20 always uses talloc, regardless of this parameter setting. This parameter must be set at the database level, and takes effect only on database restart.</p> <p>Default Value: 27 (128 MB)</p>
PatternMatchAllocator	<p>Overrides the heap memory allocator for the pattern-match library when set to 1. The Perl Compatible Regular Expressions (PCRE) pattern-match library evaluates regular expressions. Restart the database for this parameter to take effect.</p> <p>Default Value: 0</p> <p>See Regular Expression Functions.</p>
PatternMatchingUseJit	<p>Enables just-in-time compilation (to machine code) of regular expression pattern matching functions used in queries. Using this parameter can usually improve pattern matching performance on large tables. The Perl Compatible Regular Expressions (PCRE) pattern-match library evaluates regular expressions. Restart the database for this parameter to take effect.</p> <p>Default Value: 1</p> <p>See Regular Expression Functions.</p>
PcreJitStackSizeScaleFactor	<p>Determines the maximum size of the Perl Compatible Regular Expressions (PCRE) just-in-time stack. The maximum stack size will be $\text{PcreJitStackSizeScaleFactor} * 1024 * 1024$ bytes.</p>

Parameter	Description
	<p>Default Value: 32</p>
<p>PatternMatchStackAllocator</p>	<p>Overrides the stack memory allocator for the pattern-match library when set to 1. The Perl Compatible Regular Expressions (PCRE) pattern-match library evaluates regular expressions. Restart the database for this parameter to take effect.</p> <p>Default Value: 1</p> <p>See Regular Expression Functions.</p>
<p>SegmentAutoProjection</p>	<p>Determines whether auto-projections are segmented by default. Set to 0 to disable.</p> <p>Default Value: 1</p>
<p>TerraceRoutingFactor</p>	<p>Specifies a value large enough that it cannot be enabled by default, even for the largest clusters. Use the Terrace Routing equation to find the appropriate value for your cluster.</p> <p>Default Value: 1000.0</p> <p>See Terrace Routing.</p>
<p>TransactionIsolationLevel</p>	<p>Changes the isolation level for the database. After modification, Vertica uses the new transaction level for every new session. Existing sessions and their transactions continue to use the original isolation level.</p> <p>Default Value: READ COMMITTED</p> <p>See Change Transaction Isolation Levels.</p>
<p>TransactionMode</p>	<p>Specifies whether transactions are in read/write or read-only modes. Read/write is the default. Existing sessions and their transactions continue to use the original isolation level.</p>

Parameter	Description
	Default Value: READ WRITES
UDxFencedBlockTimeout	<p>Specifies the number of seconds to wait for output before aborting a UDx running in Fenced Mode. If the server aborts a UDx for this reason, it produces an error message similar to "ERROR 3399: Failure in UDx RPC call: timed out in receiving a UDx message". If you see this error frequently, you can increase this limit. UDxs running in fenced mode do not run in the server process, so increasing this value does not impede server performance.</p> <p>Default Value: 60</p>

Tuple Mover Parameters

These parameters control how the Tuple Mover operates.

Parameters	Description
ActivePartitionCount	<p>Sets the number of active partitions. The active partitions are those most recently created. For example:</p> <pre>=> ALTER DATABASE mydb SET ActivePartitionCount = 2;</pre> <p>For information about how the Tuple Mover treats active (and inactive) partitions during a mergeout operation, see Mergeout.</p> <p>Default Value: 1</p>
CancelTMTimeout	<p>When partition, copy table, and rebalance operations encounter a conflict with an internal Tuple Mover job, those operations attempt to cancel the conflicting Tuple Mover job. This parameter specifies the amount of time, in seconds, that the blocked operation waits for the Tuple Mover cancellation to take effect. If the operation is unable to cancel the Tuple Mover job within limit specified</p>

Parameters	Description
	<p>by this parameter, the operation displays an error and rolls back.</p> <p>Default Value: 300</p>
EnableTMONRecoveringNode	<p>When enabled, allows Tuple Mover to perform moveout and mergeout activities on nodes with a node state of RECOVERING. Enabling Tuple Mover reduces the number of ROS containers generated during recovery. Having fewer than 1024 ROS containers per projection allows Vertica to maintain optimal recovery performance.</p> <p>Default Value: 1</p>
MaxMrgOutROSSizeMB	<p>Specifies in MB the maximum size of ROS containers that are candidates for mergeout operations. The Tuple Mover avoids merging ROS containers that are larger than this setting.</p> <p>Default Value: -1 (no maximum limit)</p>
MergeOutInterval	<p>Specifies in seconds how long the Tuple Mover waits between checks for new ROS files to merge out. If ROS containers are added frequently, consider a value less than the default.</p> <p>Default Value: 600</p>
MoveOutInterval	<p>Specifies in seconds how long Tuple Mover waits between checks for new data in the WOS to move to ROS.</p> <p>Default Value: 300</p>
MoveOutMaxAgeTime	<p>Specifies in seconds how long the Tuple Mover waits before it is forced to write the WOS to disk.</p> <p>Default Value: 1800</p>
MoveOutSizePct	<p>The percentage of the WOS that can be filled with data before the Tuple Mover performs a moveout operation.</p> <p>Default Value: 0</p>
PurgeMergeoutPercent	<p>Specifies as a percentage the threshold of deleted records in a ROS container that invokes an automatic mergeout</p>

Parameters	Description
	<p>operation, to purge those records. Vertica only counts the number of 'aged-out' delete vectors—that is, delete vectors that are as 'old' or older than the ancient history mark (AHM) epoch.</p> <p>This threshold applies to all ROS containers for non-partitioned tables. It also applies to ROS containers of all inactive partitions. In both cases, aged-out delete vectors are permanently purged from the ROS container.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Note: This configuration parameter only applies to automatic mergeout operations. It does not apply to manual mergeout operations that are invoked by calling meta-functions <code>DO_TM_TASK('mergeout')</code> and <code>PURGE</code>.</p> </div> <p>Default Value: 20 (percent)</p>

Projection Parameters

The following configuration parameters help you manage projections.

Parameters	Description
AnalyzeRowCountInterval	<p>Specifies how often Vertica checks the number of projection rows and whether the threshold set by <code>ARCCommitPercentage</code> has been crossed.</p> <p>For more information, see Collecting Statistics.</p> <p>Default Value: 60 seconds</p>
ARCCommitPercentage	<p>Sets the threshold percentage of WOS to ROS rows, which determines when to aggregate projection row counts and commit the result to the catalog. Vertica performs this action when the WOS to ROS percentage exceeds this setting.</p> <p>Default Value: 3 (percent)</p>
ContainersPerProjectionLimit	<p>Specifies how many ROS containers Vertica creates</p>

Parameters	Description
	<p>per projection before ROS pushback occurs.</p> <p>Default Value: 1024</p> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <p>Caution: Increasing this parameter's value can cause serious degradation of database performance. Vertica strongly recommends that you not modify this parameter without first consulting with Customer Support professionals.</p> </div>
<p>EnableGroupByProjections</p>	<p>When set to 1, you can create live aggregate projections. For more information, see Live Aggregate Projections.</p> <p>Default Value: 1</p>
<p>EnableExprsInProjections</p>	<p>When set to 1, you can create projections that use expressions to calculate column values. For more information, see Aggregating Data Through Expressions.</p> <p>Default Value: 1</p>
<p>EnableTopKProjections</p>	<p>When set to 1, you can create Top-K projections that let you retrieve Top-K data quickly. For more information, see Top-K Projections.</p> <p>Default Value: 1</p>
<p>MaxAutoSegColumns</p>	<p>Specifies the number of columns (0 –1024) to segment automatically when creating auto-projections from COPY and INSERT INTO statements.</p> <p>Set to 0 to use all columns in the hash segmentation expression.</p> <p>Default Value: 32</p>
<p>RebalanceQueryStorageContainers</p>	<p>By default, prior to performing a rebalance, Vertica performs a system table query to compute the size of all projections involved in the rebalance task. This query enables Vertica to optimize the rebalance to most efficiently utilize available disk space. This</p>

Parameters	Description
	<p>query can, however, significantly increase the time required to perform the rebalance.</p> <p>By disabling the system table query, you can reduce the time required to perform the rebalance. If your nodes are low on disk space, disabling the query increases the chance that a node runs out of disk space. In that situation, the rebalance fails.</p> <p>Default Value: 1</p>
SegmentAutoProjection	<p>Determines whether auto-projections are segmented by default. Set to 0 to disable.</p>

Epoch Management Parameters

The following table describes the epoch management parameters for configuring Vertica.

Parameters	Description
AdvanceAHMInterval	<p>Determines how frequently (in seconds) Vertica checks the history retention status.</p> <p>Note: AdvanceAHMInterval cannot be set to a value that is less than the EpochMapInterval.</p> <p>Default Value: 180 (3 minutes)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET AdvanceAHMInterval = '3600';</pre>
AHMBackupManagement	<p>Blocks the advancement of the Ancient History Mark (AHM). When this parameter is enabled, the AHM epoch cannot be later than the epoch of your latest full backup. If you advance the AHM to purge and delete data, do not enable this parameter.</p> <p>Note: Do not enable this parameter before taking full backups, as it would prevent the AHM from advancing.</p> <p>Default Value: 0</p> <p>Example:</p>

Parameters	Description
	<pre>ALTER DATABASE mydb SET AHMBackupManagement = '1';</pre>
<p>EpochMapInterval</p>	<p>Determines the granularity of mapping between epochs and time available to historical queries. When a historical queries <code>AT TIME T</code> request is issued, Vertica maps it to an epoch within a granularity of EpochMapInterval seconds. It similarly affects the time reported for Last Good Epoch during Failure Recovery. Note that it does not affect internal precision of epochs themselves.</p> <p>Tip: Decreasing this interval increases the number of epochs saved on disk. Therefore, consider reducing the HistoryRetentionTime parameter to limit the number of history epochs that Vertica retains.</p> <p>Default Value: 180 (3 minutes)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EpochMapInterval = '300';</pre>
<p>HistoryRetentionTime</p>	<p>Determines how long deleted data is saved (in seconds) as an historical reference. When the specified time since the deletion has passed, you can purge the data. Use the -1 setting if you prefer to use HistoryRetentionEpochs to determine which deleted data can be purged.</p> <p>Note: The default setting of 0 effectively prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM.</p> <p>Tip: If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window to remove loaded data. For example:</p> <pre>ALTER DATABASE mydb SET HistoryRetentionTime = 86400;</pre> <p>Default Value: 0 (Data saved when nodes are down.)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HistoryRetentionTime = '240';</pre>
<p>HistoryRetentionEpochs</p>	<p>Specifies the number of historical epochs to save, and therefore, the amount of deleted data.</p>

Parameters	Description
	<p>Unless you have a reason to limit the number of epochs, Micro Focus recommends that you specify the time over which deleted data is saved.</p> <p>If you specify both <code>History</code> parameters, <code>HistoryRetentionTime</code> takes precedence. Setting both parameters to <code>-1</code>, preserves all historical data.</p> <p>See Setting a Purge Policy.</p> <p>Default Value: <code>-1</code> (Disabled)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HistoryRetentionEpochs = '40';</pre>

Monitoring Parameters

The following table describes the monitoring parameters for configuring Vertica.

Parameters	Description
<code>SnmpTrapDestinationsList</code>	<p>Defines where Vertica sends traps for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: none</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'localhost 162 public';</pre>
<code>SnmpTrapsEnabled</code>	<p>Enables event trapping for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SnmpTrapsEnabled = 1;</pre>
<code>SnmpTrapEvents</code>	<p>Define which events Vertica traps through SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: Low Disk Space, Read Only File System, Loss of K Safety, Current Fault Tolerance at Critical Level, Too Many ROS Containers, WOS Over Flow, Node State Change, Recovery</p>

Parameters	Description
	<p>Failure, Stale Checkpoint, and CRC Mismatch.</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SnmpTrapEvents = 'Low Disk Space, Recovery Failure';</pre>
SyslogEnabled	<p>Enables event trapping for syslog. See Configuring Reporting for Syslog.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SyslogEnabled = 1);</pre>
SyslogEvents	<p>Defines events that generate a syslog entry. See Configuring Reporting for Syslog.</p> <p>Default Value: none</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SyslogEvents = 'Low Disk Space, Recovery Failure';</pre>
SyslogFacility	<p>Defines which SyslogFacility Vertica uses. See Configuring Reporting for Syslog.</p> <p>Default Value: user</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SyslogFacility = 'ftp';</pre>

Profiling Parameters

The following table describes the profiling parameters for configuring Vertica. See [Profiling Database Performance](#) for more information on profiling queries.

Parameters	Description
GlobalIEEProfiling	<p>Enables profiling for query execution runs in all sessions on all nodes.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET GlobalIEEProfiling = 1;</pre>

GlobalQueryProfiling	<p>Enables query profiling for all sessions on all nodes.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET GlobalQueryProfiling = 1;</pre>
GlobalSessionProfiling	<p>Enables session profiling for all sessions on all nodes.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET GlobalSessionProfiling = 1;</pre>

Security Parameters

Use these client authentication configuration parameters and general security parameters to configure security.

Parameters	Description
DefaultIdleSessionTimeout	<p>Indicates a default session timeout value when the parameter idlesessiontimeout is not set for the user.</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET defaultidlesessiontimeout = '300 secs';</pre>
EnableAllRolesOnLogin	<p>Automatically enables all roles granted to a user once that user logs in. Enabling this eliminates the need for the user to run SET ROLE <rolenames>. Valid values are:</p> <p>0 - does not automatically enable roles</p> <p>1 - automatically enables roles</p> <p>Default Value: 0</p>
EnabledCipherSuites	<p>Indicates which SSL cipher-suites to use for secure client-server communication.</p> <p>Default Value: ALL : !ADH : !LOW : !EXP : !MD5 : !RC4 : @STRENGTH</p> <p>This setting excludes weaker cipher suites.</p> <p>Find a complete mapping of cipher suite names from JSSE to OpenSSL at openssl.org.</p>

Parameters	Description
EnableSSL	<p>Enables SSL for the server. See Implementing SSL.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EnableSSL = '1';</pre>
GlobalHeirUserName	<p>The user name which inherits objects previously owned by dropped users (using CASCADE). This prevents the loss of data owned by dropped users. If the user name indicated here does not exist, the system automatically creates the user.</p> <p>Valid Values:</p> <p><auto> (default) - re-parents objects to the dbadmin user by default.</p> <p>Note: When setting to the default, include the angle brackets < >.</p> <p><username> - re-parents objects to the username you enter.</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET PARAMETER GlobalHeirUsername='userheir1';</pre> <p>If you do not set this parameter, the objects of dropped users do not get re-parented.</p>
RequireFIPS	<p>This parameter indicates whether the FIPS mode is enabled or disabled. Upon startup Vertica automatically sets this parameter, and you cannot modify it.</p> <p>0 - Disabled FIPS</p> <p>1 - Enabled FIPS</p> <p>The value of this parameter matches the contents of the file, crypto.fips_enabled. See Implement FIPS on the Server.</p>
RestrictSystemTables	<p>Prohibits non-database administrator users from viewing sensitive information in system tables. Valid values are:</p> <p>0 - Allows all users to access system tables</p> <p>1 — Limits access to system tables to database administrator users</p>

Parameters	Description
	<p>Default Value: 0</p> <p>See System Table Restriction.</p>
SecurityAlgorithm	<p>Sets the algorithm for the function that hash authentication uses MD5 or SHA-512.</p> <p>Default Value: 'NONE '</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SecurityAlgorithm = 'SHA512';</pre>
SSLCA	<p>Sets the SSL certificate authority.</p> <p>Default Value: No default value</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SSLCA = '<contents of certificate authority root.crt file>';</pre> <p>Include the contents of the certificate authority, root . crt, file, but do not include the file name.</p>
SSLCertificate	<p>Sets the SSL certificate. If your SSL certificate is a certificate chain, cut and paste only the top-most certificate of the certificate chain to set this value.</p> <p>Default Value: No default value</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SSLCertificate = '<contents of server.crt file>';</pre> <p>Include the contents of the server . crt file, but do not include the file name.</p> <p>Note: This parameter gets set automatically during upgrade to 7.1 if you set EnableSSL=1 prior to the upgrade.</p>
SSLPrivateKey	<p>Specifies the server's private key. The value of this parameter is visible only to dbadmin users.</p> <p>Default Value: No default value</p>

Parameters	Description
	<p>Example:</p> <pre>ALTER DATABASE mydb SET SSLPrivateKey = '<contents of server.key file>';</pre> <p>Include the contents of the server .key file, but do not include the file name.</p> <p>Note: This parameter gets set automatically during upgrade to 7.1 if you set EnableSSL=1 prior to the upgrade.</p>

View parameter values with the statement, `SHOW DATABASE`. You must be a database superuser to view the value:

```
SHOW DATABASE mydb SSLCertificate;
```

See Also

[Kerberos Authentication Parameters](#)

[Configuring SSL](#)

Database Designer Parameters

The following table describes the parameters for configuring the Vertica Database Designer.

Parameter	Description
<code>DBDCorrelationSampleRowCount</code>	<p>Minimum number of table rows at which Database Designer discovers and records correlated columns.</p> <p>Default Value: 4000</p>
<code>DBDLogInternalDesignProcess</code>	<p>Enables or disables Database Designer logging.</p> <p>Default value: 0 (False)</p>
<code>DBDUseOnlyDesignerResourcePool</code>	<p>Enables use of the DBD pool by the Vertica Database Designer.</p> <p>When set to false, design processing is mostly</p>

Parameter	Description
	<p>contained by the user's resource pool, but might spill over into some system resource pools for less-intensive tasks</p> <p>Default value: 0 (False)</p>

Internationalization Parameters

The following table describes the internationalization parameters for configuring Vertica.

Parameters	Description
DefaultIntervalStyle	<p>Sets the default interval style to use. If set to 0 (default), the interval is in PLAIN style (the SQL standard), no interval units on output. If set to 1, the interval is in UNITS on output. This parameter does not take effect until the database is restarted.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET DefaultIntervalStyle = 1;</pre>
DefaultSessionLocale	<p>Sets the default session startup locale for the database. This parameter does not take effect until the database is restarted.</p> <p>Default Value: en_US@collation=binary</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET DefaultSessionLocale = 'en_GB';</pre>
EscapeStringWarning	<p>Issues a warning when back slashes are used in a string literal. This is provided to help locate back slashes that are being treated as escape characters so they can be fixed to follow the Standard conforming string syntax instead.</p> <p>Default Value: 1</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EscapeStringWarning = '1';</pre>
StandardConformingStrings	<p>Determines whether ordinary string literals ('...') treat</p>

Parameters	Description
	<p>backslashes (\) as string literals or escape characters. When set to '1', backslashes are treated as string literals, when set to '0', back slashes are treated as escape characters.</p> <p>Tip: To treat backslashes as escape characters, use the Extended string syntax:</p> <pre>(E ' . . . ');</pre> <p>See String Literals (Character) in the SQL Reference Manual.</p> <p>Default Value: 1</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET StandardConformingStrings = '0';</pre>

Data Collector Parameters

The following table lists the Data Collector parameter for configuring Vertica.

Parameter	Description
EnableDataCollector	<p>Enables and disables the Data Collector, which is the Workload Analyzer's internal diagnostics utility. Affects all sessions on all nodes. Use 0 to turn off data collection.</p> <p>Default value: 1 (Enabled)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EnableDataCollector = 0;</pre>

For more information, see the following topics in the SQL Reference Manual:

- [Data Collector Functions](#)
- [ANALYZE_WORKLOAD](#)
- [V_MONITOR.DATA_COLLECTOR](#)
- [V_MONITOR.TUNING_RECOMMENDATIONS](#)

See also the following topics in the Administrator's Guide

- [Retaining Monitoring Information](#)
- [Analyzing Workloads](#)
- [Tuning Recommendations](#)
- [Analyzing Workloads Through Management Console](#) and [Through an API](#)

Text Search Parameters

You can configure Vertica for text search using these parameters.

Parameters	Description
TextIndexMaxTokenLength	<p>Controls the maximum size of a token in a text index. If the parameter is set to a value greater than 65000 characters, then the tokenizer truncates the token at 65000 characters.</p> <p>You should avoid setting this parameter near 65000 (the maximum value). Doing so can result in a significant decrease in performance. For optimal performance, the parameter should be set to the maximum token value of your tokenizer.</p> <p>Default Value: 128 characters</p> <p>Example:</p> <pre>ALTER DATABASE database_name SET PARAMETER TextIndexMaxTokenLength=760;</pre>

Kerberos Authentication Parameters

The following parameters let you configure the Vertica principal for Kerberos authentication and specify the location of the Kerberos keytab file.

Parameter	Description
KerberosServiceName	<p>Provides the service name portion of the Vertica Kerberos principal. By default, this parameter is 'vertica'. For example: vertica/host@EXAMPLE.COM.</p>
KerberosHostname	<p>[Optional] Provides the instance or host name portion of the Vertica Kerberos principal. For example: vertica/host@EXAMPLE.COM</p>

Parameter	Description
	<p>If you omit the optional KerberosHostname parameter, Vertica uses the return value from the <code>gethostname()</code> function. Assuming each cluster node has a different host name, those nodes will each have a different principal, which you must manage in that node's keytab file.</p>
KerberosRealm	<p>Provides the realm portion of the Vertica Kerberos principal. A realm is the authentication administrative domain and is usually formed in uppercase letters; for example: <code>vertica/host@EXAMPLE.COM</code>.</p>
KerberosKeytabFile	<p>Provides the location of the keytab file that contains credentials for the Vertica Kerberos principal. By default, this file is located in <code>/etc</code>. For example: <code>KerberosKeytabFile=/etc/krb5.keytab</code>.</p> <p>Notes:</p> <ul style="list-style-type: none"> The principal must take the form <code>KerberosServiceName/KerberosHostName@KerberosRealm</code> The keytab file must be readable by the file owner who is running the process (typically the Linux <code>dbadmin</code> user assigned file permissions <code>0600</code>).

Hadoop Parameters

The following table describes the general parameters for configuring integration with Apache Hadoop. See [Integrating with Apache Hadoop](#) for more information.

Parameter	Description
HadoopConfDir	<p>A directory path containing the XML configuration files copied from Hadoop. The same path must be valid on every Vertica node. You can use the <code>VERIFY_HADOOP_CONF_DIR</code> meta-function to test that the value is set correctly. Setting this parameter is required to read data from HDFS.</p> <p>When you set this parameter, any previously-cached configuration information is flushed.</p>

	<p>Default Value: obtained from environment if possible</p> <p>Requires Restart: No</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HadoopConfDir = '/hadoop/hcat/conf';</pre>
--	--

The following table describes the parameters for configuring the HCatalog Connector. See [Using the HCatalog Connector](#) in Integrating with Apache Hadoop for more information.

Parameter	Description
HCatalogConnectorUseHiveServer2	<p>When enabled, Vertica internally uses HiveServer2 instead of WebHCat to get metadata from Hive.</p> <p>Default Value: 1 (enabled)</p> <p>Requires Restart: No</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HCatalogConnectorUseHiveServer2 = 0;</pre>
HCatalogConnectorUseLibHDFSPP	<p>Whether the HCatalog Connector should use the hdfs scheme instead of webhdfs to read native formats. Using the hdfs scheme requires additional configuration but can have better performance.</p> <p>Default Value: 1 (enabled)</p> <p>Requires Restart: No</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HCatalogConnectorUseLibHDFSPP = 1;</pre>
HCatConnectionTimeout	<p>The number of seconds the HCatalog Connector waits for a successful connection to the HiveServer2 (or WebHCat) server before returning a timeout error.</p> <p>Default Value: 0 (Wait indefinitely)</p> <p>Requires Restart: No</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HCatConnectionTimeout = 30;</pre>
HCatSlowTransferLimit	<p>The lowest transfer speed (in bytes per second) that the HCatalog Connector allows when retrieving data from the HiveServer2 (or WebHCat) server. In some cases, the data transfer rate from the server to</p>

	<p>Vertica is below this threshold. In such cases, after the number of seconds specified in the HCatSlowTransferTime parameter pass, the HCatalog Connector cancels the query and closes the connection.</p> <p>Default Value: 65536</p> <p>Requires Restart: No</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HCatSlowTransferLimit = 32000;</pre>
HCatSlowTransferTime	<p>The number of seconds the HCatalog Connector waits before testing whether the data transfer from the server is too slow. See the HCatSlowTransferLimit parameter.</p> <p>Default Value: 60</p> <p>Requires Restart: No</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET HCatSlowTransferTime = 90;</pre>

Note: You can override the HCatalog configuration parameters when creating an HCatalog schema. See [CREATE HCATALOG SCHEMA](#) in the SQL Reference Manual for an explanation.

User-Defined Session Parameters

Use the following Vertica use-defined session parameters to configure Kafka SSL when not using a scheduler.

The kafka_ parameters configure SSL authentication for Kafka. Refer to [Using SSL with Kafka](#) for more information.

Parameter	Description
kafka_SSL_CA	<p>The contents of the certificate authority certificate.</p> <p>Default Value: none</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER kafka_SSL_CA='MIIBOQIBAAJBAIOL';</pre>

Parameter	Description
kafka_SSL_Certificate	<p>The contents of the SSL certificate.</p> <p>Default Value: none</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER kafka_SSL_Certificate='XrM0704dV/nJ5g';</pre>
kafka_SSL_PrivateKey_secret	<p>The private key used to encrypt the session. Vertica does not log this information.</p> <p>Default Value: none</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKey_secret='A60iThKtezaCk7F';</pre>
kafka_SSL_PrivateKeyPassword_secret	<p>The password used to create the private key. Vertica does not log this information.</p> <p>Default Value: none</p> <p>Example:</p> <pre>ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKeyPassword_secret='secret';</pre>
kafka_Enable_SSL	<p>Enables SSL authentication for Vertica-Kafka integration.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>=> ALTER SESSION SET kafka_Enable_SSL=1;</pre>
MaxSessionUDParameterSize	<p>Sets the maximum length of a value in a user-defined session parameter.</p> <p>Default Value: 1000</p> <p>Example:</p> <pre>=> ALTER SESSION SET MaxSessionUDParameterSize = 2000</pre>

Related Topics

- [User-Defined Session Parameters](#)

Constraint Enforcement Parameters

The following configuration parameters enforce constraints.

Use the [ALTER DATABASE](#) statement to set these parameters. You do not need to restart your database after setting them.

- The parameter settings apply for any check constraint, and any primary or unique key constraint that you have not explicitly enabled or disabled within a [CREATE TABLE](#) or [ALTER TABLE](#) statement.
- Any new check constraint, and any primary or unique key constraint that you create or alter is set according to the value of the corresponding parameter unless you specifically enabled or disabled the constraint.

Important: Setting a constraint as enabled or disabled when you create or alter it using [CREATE TABLE](#) or [ALTER TABLE](#) overrides the parameter setting.

Parameters	Description
EnableNewCheckConstraintsByDefault	<p>Set to 1 (the default) to automatically enable newly created check constraints that you specified through CREATE TABLE or ALTER TABLE statements. However, if you have explicitly disabled a constraint when you created or altered it, it is not enforced.</p> <p>Default Value: 1 (Enabled)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EnableNewCheckConstraintsByDefault = 0;</pre>
EnableNewPrimaryKeysByDefault	<p>Set to 1 to automatically enable newly created primary key constraints that you specified through CREATE TABLE or ALTER TABLE statements. However, if you have explicitly disabled a constraint when you created or altered it, it is not enforced.</p>

Parameters	Description
	<p>Default Value: 0 (Disabled)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EnableNewPrimaryKeysByDefault = 1;</pre>
EnableNewUniqueKeysByDefault	<p>Set to 1 to automatically enable newly created unique constraints that you specified through CREATE TABLE or ALTER TABLE statements. However, if you have explicitly disabled a constraint when you created or altered it, it is not enforced.</p> <p>Default Value: 0 (Disabled)</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET EnableNewUniqueKeysByDefault = 1;</pre>

Note: Vertica recommends enabling primary key enforcement if you have enabled unique key enforcement.

Numeric Precision Parameters

The following configuration parameters let you configure numeric precision for numeric data types. For more about using these parameters, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

Parameters	Description
AllowNumericOverflow	<p>When set to 1 (True), allows silent numeric overflow. When true, Vertica does not implicitly extend precision of numeric data types.</p> <p>Default Value: 1 (True)</p> <p>When set to 0 (False), Vertica produces an overflow error, if a result exceeds the precision set by NumericSumExtraPrecisionDigits.</p> <p>Vertica ignores the value of NumericSumExtraPrecisionDigits when</p>

Parameters	Description
	<p>AllowNumericOverflow is true.</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET PARAMETER AllowNumericOverflow=0;</pre>
NumericSumExtraPrecisionDigits	<p>Vertica produces an overflow error, if a result exceeds the specified precision, By default this specified precision is six places beyond the DDL-specified precision.</p> <p>Default Value: 6</p> <p>This parameter setting only applies if AllowNumericOverflow is set to 0 (False).</p> <p>Possible Values: 0 to 20.</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET PARAMETER NumericSumExtraPrecisionDigits=8</pre>

Vertica Library for Amazon Web Services Parameters

Use these parameters to configure the Verticalibrary for Amazon Web Services (AWS). All parameters listed are case sensitive.

Parameter	Description
aws_id	<p>Your AWS access key ID.</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_id='<YOUR AWS ID>';</pre>
aws_secret	<p>Your AWS secret access key.</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_secret='<YOUR AWS KEY>';</pre>
aws_region	<p>The region your S3 bucket is located. aws_region can only be configured with</p>

Parameter	Description
	<p>one region at a time. If you need to access buckets in multiple regions, you must re-set the parameter each time you change regions.</p> <p>Default value: us-east-1</p> <p>You can find more information about AWS regions in the Amazon Documentation.</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_region='<REGION IDENTIFIER>';</pre>
aws_ca_path	<p>The path which Vertica will use when looking for SSL server certificates.</p> <p>Default value: system dependent</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_ca_path= /home/user/ssl_folder;</pre>
aws_ca_bundle	<p>The path which Vertica will use when looking for a SSL server certificate bundle.</p> <p>Default value: system dependent</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_ca_bundle= /home/user/ssl_folder/ca_bundle;</pre>
aws_proxy	<p>A string value which allows you to set an HTTP/HTTPS proxy for the AWS library.</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_proxy='192.168.1.1:8080;</pre>
aws_verbose	<p>When enabled, logs libcurl debug messages to dbLog.</p> <p>Default value: false</p> <p>Example:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_verbose= 1;</pre>
aws_max_send_	<p>Sets the maximum transfer speed when sending data to AWS S3 in bytes per second.</p>

Parameter	Description
speed	<p>Example:</p> <p>The following example sets a maximum send speed of 1KB/S:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_max_send_speed= 1024;</pre>
aws_max_recv_speed	<p>Sets the maximum transfer speed when receiving data to AWS S3 in bytes per second.</p> <p>Example:</p> <p>The following example sets a maximum receive speed of 100KB/S:</p> <pre>=> ALTER SESSION SET UDPARAMETER FOR awslib aws_max_recv_speed= 102400;</pre>

For more information, see the following topics in the Administrator's Guide:

- [AWS Library](#)
- [Configuring Vertica AWS Library](#)
- [Export AWS Library](#)
- [Import AWS Library](#)

Designing a Logical Schema

Designing a logical schema for a Vertica database is the same as designing for any other SQL database. A logical schema consists of objects such as schemas, tables, views and referential Integrity constraints that are visible to SQL users. Vertica supports any relational schema design that you choose.

Using Multiple Schemas

Using a single schema is effective if there is only one database user or if a few users cooperate in sharing the database. In many cases, however, it makes sense to use additional schemas to allow users and their applications to create and access tables in separate namespaces. For example, using additional schemas allows:

- Many users to access the database without interfering with one another.

Individual schemas can be configured to grant specific users access to the schema and its tables while restricting others.

- Third-party applications to create tables that have the same name in different schemas, preventing table collisions.

Unlike other RDBMS, a schema in an Vertica database is not a collection of objects bound to one user.

Multiple Schema Examples

This section provides examples of when and how you might want to use multiple schemas to separate database users. These examples fall into two categories: using multiple private schemas and using a combination of private schemas (i.e. schemas limited to a single user) and shared schemas (i.e. schemas shared across multiple users).

Using Multiple Private Schemas

Using multiple private schemas is an effective way of separating database users from one another when sensitive information is involved. Typically a user is granted access to only one schema and its contents, thus providing database security at the schema level. Database users can be running different applications, multiple copies of the same application, or even multiple instances of the same application. This enables you to consolidate applications on one database to reduce management overhead and use resources more effectively. The following examples highlight using multiple private schemas.

Using multiple schemas to separate users and their unique applications

In this example, both database users work for the same company. One user (HRUser) uses a Human Resource (HR) application with access to sensitive personal data, such as salaries, while another user (MedUser) accesses information regarding company healthcare costs through a

healthcare management application. HRUser should not be able to access company healthcare cost information and MedUser should not be able to view personal employee data.

To grant these users access to data they need while restricting them from data they should not see, two schemas are created with appropriate user access, as follows:

- HRSchema—A schema owned by HRUser that is accessed by the HR application.
- HealthSchema—A schema owned by MedUser that is accessed by the healthcare management application.

Using multiple schemas to support multitenancy

This example is similar to the last example in that access to sensitive data is limited by separating users into different schemas. In this case, however, each user is using a virtual instance of the same application.

An example of this is a retail marketing analytics company that provides data and software as a service (SaaS) to large retailers to help them determine which promotional methods they use are most effective at driving customer sales.

In this example, each database user equates to a retailer, and each user only has access to its own schema. The retail marketing analytics company provides a virtual instance of the same application to each retail customer, and each instance points to the user's specific schema in which to create and update tables. The tables in these schemas use the same names because they are created by instances of the same application, but they do not conflict because they are in separate schemas.

Example of schemas in this database could be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.

Using multiple schemas to migrate to a newer version of an application

Using multiple schemas is an effective way of migrating to a new version of a software application. In this case, a new schema is created to support the new version of the software, and the old schema is kept as long as necessary to support the original version of the software. This is called a “rolling application upgrade.”

For example, a company might use a HR application to store employee data. The following schemas could be used for the original and updated versions of the software:

- HRSchema—A schema owned by HRUser, the schema user for the original HR application.
- V2HRSchema—A schema owned by V2HRUser, the schema user for the new version of the HR application.

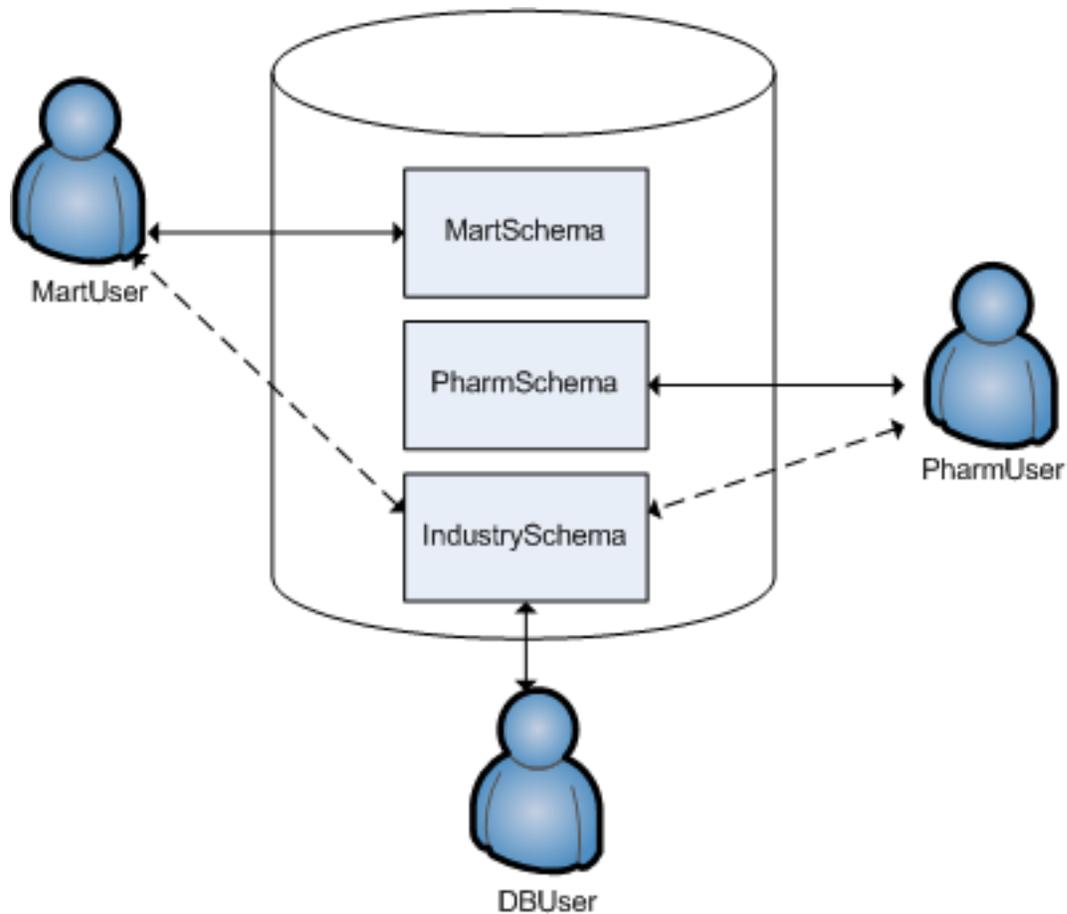
Combining Private and Shared Schemas

The previous examples illustrate cases in which all schemas in the database are private and no information is shared between users. However, users might want to share common data. In the retail case, for example, MartUser and PharmUser might want to compare their per store sales of a particular product against the industry per store sales average. Since this information is an industry average and is not specific to any retail chain, it can be placed in a schema on which both users are granted USAGE privileges. (For more information about schema privileges, see [Schema Privileges](#).)

Example of schemas in this database might be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.
- IndustrySchema—A schema owned by DBUser (from the retail marketing analytics company) on which both MartUser and PharmUser have USAGE privileges. It is unlikely that retailers would be given any privileges beyond USAGE on the schema and SELECT on one or

more of its tables.



Creating Schemas

You can create as many schemas as necessary for your database. For example, you could create a schema for each database user. However, schemas and users are not synonymous as they are in Oracle.

By default, only a superuser can create a schema or give a user the right to create a schema. (See [GRANT \(Database\)](#) in the SQL Reference Manual.)

To create a schema use the [CREATE SCHEMA](#) statement, as described in the SQL Reference Manual.

Specifying Objects in Multiple Schemas

Once you create two or more schemas, each SQL statement or function must identify the schema associated with the object you are referencing. You can specify an object within multiple schemas by:

- Qualifying the object name by using the schema name and object name separated by a dot. For example, to specify `MyTable`, located in `Schema1`, qualify the name as `Schema1.MyTable`.
- Using a search path that includes the desired schemas when a referenced object is unqualified. By [Setting Search Paths](#), Vertica will automatically search the specified schemas to find the object.

Setting Search Paths

Each user session has a search path of schemas. Vertica uses this search path to find tables and user-defined functions (UDFs) that are unqualified by their schema name. A session search path is initially set from the user's profile. You can change the session's search path at any time by calling `SET SEARCH_PATH`. This search path remains in effect until the next `SET SEARCH_PATH` statement, or the session ends.

Viewing the Current Search Path

`SHOW SEARCH_PATH` returns the session's current search path. For example:

```
=> SHOW SEARCH_PATH;
  name      |                setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
```

Schemas are listed in descending order of precedence. The first schema has the highest precedence in the search order. If this schema exists, it is also defined as the current schema, which is used for tables that are created with unqualified names. You can identify the current schema by calling the function `CURRENT_SCHEMA`:

```
=> SELECT CURRENT_SCHEMA;
 current_schema
-----
public
(1 row)
```

Setting the User Search Path

A session search path is initially set from the user's profile. If the search path in a user profile is not set by [CREATE USER](#) or [ALTER USER](#), it is set to the database default:

```
=> CREATE USER agent007;
CREATE USER
=> \c - agent007
You are now connected as user "agent007".
=> SHOW SEARCH_PATH;
  name      |                               setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
```

`$user` resolves to the session user name—in this case, `agent007`—and has the highest precedence. If a schema `agent007`, exists, Vertica begins searches for unqualified tables in that schema. Also, calls to [CURRENT_SCHEMA](#) return this schema. Otherwise, Vertica uses `public` as the current schema and begins searches in it.

Use [ALTER USER](#) to modify an existing user's search path. These changes overwrite all non-system schemas in the search path, including `$USER`. System schemas are untouched. Changes to a user's search path take effect only when the user starts a new session; current sessions are unaffected.

Important: After modifying the user's search path, [verify that the user has access privileges to all schemas that are on the updated search path.](#)

For example, the following statements modify `agent007`'s search path, and grant access privileges to schemas and tables that are on the new search path:

```
=> ALTER USER agent007 SEARCH_PATH store, public;
ALTER USER
=> GRANT ALL ON SCHEMA store, public TO agent007;
GRANT PRIVILEGE
=> GRANT SELECT ON ALL TABLES IN SCHEMA store, public TO agent007;
GRANT PRIVILEGE
=> \c - agent007
You are now connected as user "agent007".
=> SHOW SEARCH_PATH;
  name      |                               setting
-----+-----
search_path | store, public, v_catalog, v_monitor, v_internal
(1 row)
```

To verify a user's search path, query the system table [USERS](#):

```
=> SELECT search_path FROM USERS WHERE user_name='agent007';
      search_path
-----
```

```
store, public, v_catalog, v_monitor, v_internal  
(1 row)
```

To revert a user's search path to the database default settings, call `ALTER USER` and set the search path to `DEFAULT`. For example:

```
=> ALTER USER agent007 SEARCH_PATH DEFAULT;  
ALTER USER  
=> SELECT search_path FROM USERS WHERE user_name='agent007';  
          search_path  
-----  
"$user", public, v_catalog, v_monitor, v_internal  
(1 row)
```

Ignored Search Path Schemas

Vertica only searches among existing schemas to which the current user has access privileges. If a schema in the search path does not exist or the user lacks access privileges to it, Vertica silently excludes it from the search. For example, if `agent007` lacks `SELECT` privileges to schema `public`, Vertica silently skips this schema. Vertica returns with an error only if it cannot find the table anywhere on the search path.

Setting Session Search Path

Vertica initially sets a session's search path from the user's profile. You can change the current session's search path with [SET SEARCH_PATH](#). You can use `SET SEARCH_PATH` in two ways:

- Explicitly set the session search path to one or more schemas. For example:

```
=> \c - agent007  
You are now connected as user "agent007".  
dbadmin=> SHOW SEARCH_PATH;  
  name | setting  
-----+-----  
search_path | "$user", public, v_catalog, v_monitor, v_internal  
(1 row)  
  
=> SET SEARCH_PATH TO store, public;  
SET  
=> SHOW SEARCH_PATH;  
  name | setting  
-----+-----  
search_path | store, public, v_catalog, v_monitor, v_internal  
(1 row)
```

- Set the session search path to the database default:

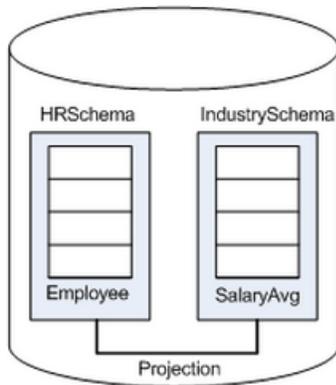
```
=> SET SEARCH_PATH TO DEFAULT;
SET
=> SHOW SEARCH_PATH;
  name | setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

SET SEARCH_PATH overwrites all non-system schemas in the search path, including \$USER. System schemas are untouched.

Creating Objects That Span Multiple Schemas

Vertica supports views that reference tables across multiple schemas. For example, a user might need to compare employee salaries to industry averages. In this case, the application queries two schemas:

- Shared schema IndustrySchema for salary averages
- Private schema HRSchema for company-specific salary information



Best Practice: When creating objects that span schemas, use qualified table names. This naming convention avoids confusion if the query path or table structure within the schemas changes at a later date.

Tables in Schemas

In Vertica you can create persistent and temporary tables, through [CREATE TABLE](#) and [CREATE TEMPORARY TABLE](#), respectively.

For detailed information on both types, see [Creating Tables](#) and [Creating Temporary Tables](#).

Persistent Tables

CREATE TABLE creates a table in the Vertica logical schema. For example:

```
CREATE TABLE vendor_dimension (  
  vendor_key      INTEGER      NOT NULL PRIMARY KEY,  
  vendor_name     VARCHAR(64),  
  vendor_address  VARCHAR(64),  
  vendor_city     VARCHAR(64),  
  vendor_state    CHAR(2),  
  vendor_region   VARCHAR(32),  
  deal_size       INTEGER,  
  last_deal_update DATE  
);
```

For detailed information, see [Creating Tables](#).

Temporary Tables

CREATE TEMPORARY TABLE creates a table whose data persists only during the current session. Temporary table data is never visible to other sessions.

Temporary tables can be used to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated—for example, the tool first gets a result set, then queries the result set, and so on.

CREATE TEMPORARY TABLE can create tables at two scopes, global and local, through the keywords **GLOBAL** and **LOCAL**, respectively:

- **GLOBAL** (default): The table definition is visible to all sessions. However, table data is session-scoped.
- **LOCAL**: The table definition is visible only to the session in which it is created. When the session ends, Vertica automatically drops the table.

For detailed information, see [Creating Temporary Tables](#).

Creating a Database Design

A *design* is a physical storage plan that optimizes query performance. Data in Vertica is physically stored in projections. When you initially load data into a table using `INSERT`, `COPY` (or `COPY LOCAL`), Vertica creates a default superprojection for the table. This superprojection ensures that all of the data is available for queries. However, these superprojections might not optimize database performance, resulting in slow query performance and low data compression.

To improve performance, create a design for your Vertica database that optimizes query performance and data compression. You can create a design in several ways:

- [Use Database Designer](#), a tool that recommends a design for optimal performance.
- [Manually create a design](#)
- Use Database Designer to create an initial design and then manually modify it.

Database Designer can help you minimize how much time you spend on manual database tuning. You can also use Database Designer to redesign the database incrementally as requirements such as workloads change over time.

Database Designer runs as a background process. This is useful if you have a large design that you want to run overnight. An active SSH session is not required, so design and deploy operations continue to run uninterrupted if the session ends.

Tip: Micro Focus recommends that you first globally optimize your database using the Comprehensive setting in Database Designer. If the performance of the comprehensive design is not adequate, you can design custom projections using an incremental design and manually, as described in [Creating Custom Designs](#).

About Database Designer

Vertica Database Designer uses sophisticated strategies to create a design that provides excellent performance for ad-hoc queries and specific queries while using disk space efficiently.

During the design process, Database Designer analyzes the logical schema definition, sample data, and sample queries, and creates a physical schema (projections) in the form of a SQL script that you deploy automatically or manually. This script creates a minimal set of superprojections to ensure K-safety.

In most cases, the projections that Database Designer creates provide excellent query performance within physical constraints while using disk space efficiently.

General Design Options

When you run Database Designer, several general options are available:

- Create a comprehensive or incremental design.
- Optimize for query execution, load, or a balance of both.
- Require K-safety.
- Recommend unsegmented projections when feasible.
- Analyze statistics before creating the design.

Design Input

Database Designer bases its design on the following information that you provide:

- **Design queries** that you typically run during normal database operations.
- **Design tables** that contain sample data.

Output

Database Designer yields the following output:

- A design script that creates the projections for the design in a way that meets the optimization objectives and distributes data uniformly across the cluster.
- A deployment script that creates and refreshes the projections for your design. For comprehensive designs, the deployment script contains commands that remove non-optimized projections. The deployment script includes the full design script.
- A backup script that contains SQL statements to deploy the design that existed on the system before deployment. This file is useful in case you need to revert to the pre-deployment design.

Design Restrictions

Database Designer-generated designs:

- Exclude live aggregate or Top-K projections. You must create these manually. See [CREATE PROJECTION \(Live Aggregate Projections\)](#).
- Do not sort, segment, or partition projections on LONG VARBINARY and LONG VARCHAR columns.

Post-Design Options

While running Database Designer, you can choose to deploy your design automatically after the deployment script is created, or to deploy it manually, after you have reviewed and tested the design. Vertica recommends that you test the design on a non-production server before deploying the design to your production server.

How Database Designer Creates a Design

Design Recommendations

Database Designer-generated designs can include the following recommendations:

- Sort buddy projections in the same order, which can significantly improve load, recovery, and site node performance. All buddy projections have the same base name so that they can be identified as a group.

Note: If you manually create projections, Database Designer recommends a buddy with the same sort order, if one does not already exist. By default, Database Designer recommends both super and non-super segmented projections with a buddy of the same sort order and segmentation.

- Accepts unlimited queries for a comprehensive design.
- Allows you to analyze column correlations. Correlation analysis typically only needs to be performed once and only if the table has more than DBDCorrelationSampleRowCount

(default: 4000) rows.

By default, Database Designer does not analyze column correlations. To set the correlation analysis mode, use [DESIGNER_SET_ANALYZE_CORRELATIONS_MODE](#)

- Identifies similar design queries and assigns them a signature.

For queries with the same signature, Database Designer weights the queries, depending on how many queries have that signature. It then considers the weighted query when creating a design.

- Recommends and creates projections in a way that minimizes data skew by distributing data uniformly across the cluster.
- Produces higher quality designs by considering UPDATE, DELETE, and SELECT statements.

Who Can Run Database Designer

To use Administration Tools to run Database Designer and create an optimal database design, you must be a DBADMIN user.

To run Database Designer programmatically or using Management Console, you must be one of two types of users:

- DBADMIN user
- Have been assigned the DBDUSER role and be the owner of database tables for which you are creating a design

Granting and Enabling the DBDUSER Role

For a non-DBADMIN user to be able to run Database Designer using Management Console, follow the steps described in [Allowing the DBDUSER to Run Database Designer Using Management Console](#).

For a non-DBADMIN user to be able to run Database Designer programmatically, following the steps described in [Allowing the DBDUSER to Run Database Designer Programmatically](#).

Important: When you grant the DBDUSER role, make sure to associate a resource pool with that user to manage resources during Database Designer runs. (For instructions about how to associate a resource pool with a user, see [User Profiles](#).)

Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. When a user runs Database Designer, either using the Management Console or programmatically, its execution is mostly contained by the user's resource pool, but may spill over into system resource pools for less-intensive tasks.

Allowing the DBDUSER to Run Database Designer Using Management Console

To allow a user with the DBDUSER role to run Database Designer using Management Console, you first need to create the user on the Vertica server.

As DBADMIN, take these steps on the server:

1. Add a temporary folder to all cluster nodes.

```
=> CREATE LOCATION '/tmp/dbd' ALL NODES;
```

2. Create the user who needs access to Database Designer.

```
=> CREATE USER new_user;
```

3. Grant the user the privilege to create schemas on the database for which they want to create a design.

```
=> GRANT CREATE ON DATABASE new_database TO new_user;
```

4. Grant the DBDUSER role to the new user.

```
=> GRANT DBDUSER TO new_user;
```

5. On all nodes in the cluster, grant the user access to the temporary folder.

```
=> GRANT ALL ON LOCATION '/tmp/dbd' TO new_user;
```

6. Grant the new user access to the database schema and its tables.

```
=> GRANT ALL ON SCHEMA user_schema TO new_user;  
=> GRANT ALL ON ALL TABLES IN SCHEMA user_schema TO new_user;
```

After you have completed this task, you need to do the following to map the MC user to the new_user you created in the previous steps:

1. Log in to Management Console as an MC Super user.
2. Click **MC Settings**.
3. Click **User Management**.
4. To create a new MC user, click **Add**. To use an existing MC user, select the user and click **Edit**.
5. Next to the **DB access level** window, click **Add**.
6. In the **Add Permissions** window, do the following:
 - a. From the **Choose a database** drop-down list, select the database for which you want the user to be able to create a design.
 - b. In the **Database username** field, enter the user name you created on the Vertica server, `new_user` in this example.
 - c. In the Database password field, enter the password for the database you selected in step a.
 - d. In the **Restrict access** drop-down list, select the level of MC user you want for this user.
7. Click **OK** to save your changes.
8. Log out of the MC Super user account.

The MC user is now mapped to the user that you created on the Vertica server. Log in as the MC user and use Database Designer to create an optimized design for your database.

For more information about mapping MC users, see [Mapping an MC User to a Database user's Privileges](#).

Allowing the DBDUSER to Run Database Designer Programmatically

To allow a user with the DBDUSER role to run Database Designer programmatically, take these steps:

1. The DBADMIN user must grant the DBDUSER role:

```
=> GRANT DBDUSER TO <username>;
```

This role persists until the DBADMIN user revokes it.

- For a non-DBADMIN user to run the Database Designer programmatically or using Management Console, one of the following two steps must happen first:
 - If the user's default role is already DBDUSER, skip this step. Otherwise, The user must enable the DBDUSER role:

```
=> SET ROLE DBDUSER;
```

- The DBADMIN must add DBDUSER as the default role for that user:

```
=> ALTER USER <username> DEFAULT ROLE DBDUSER;
```

DBDUSER Capabilities and Limitations

The DBDUSER role has the following capabilities and limitations:

- A DBDUSER cannot create a design with a K-safety less than the system K-safety. If the designs violate the current K-safety by not having enough buddy projections for the tables, the design does not complete.
- A DBDUSER cannot explicitly change the ancient history mark (AHM), even during deployment of their design.

When you create a design, you automatically have privileges to manipulate the design. Other tasks may require that the DBDUSER have additional privileges:

To...	DBDUSER must have...
Add design tables	<ul style="list-style-type: none">USAGE privilege on the design table schemaOWNER privilege on the design table
Add a single design query	<ul style="list-style-type: none">Privilege to execute the design query
Add a file of design queries	<ul style="list-style-type: none">Read privilege on the storage location that contains the query filePrivilege to execute all the queries in the file
Add design queries from the result of a user query	<ul style="list-style-type: none">Privilege to execute the user queryPrivilege to execute each design query retrieved from the results of the user query

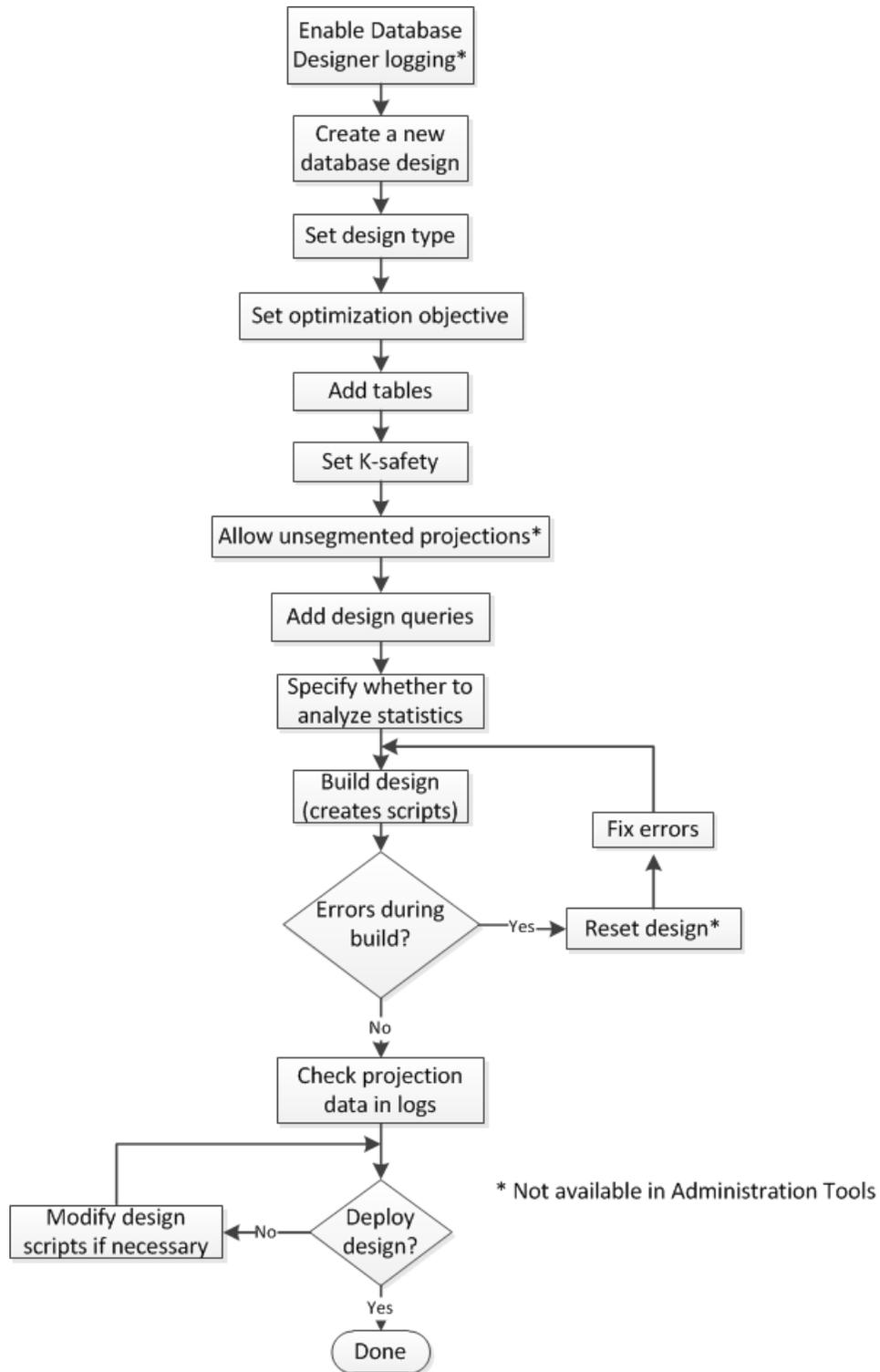
To...	DBDUSER must have...
Create the design and deployment scripts	<ul style="list-style-type: none">• WRITE privilege on the storage location of the design script• WRITE privilege on the storage location of the deployment script

Workflow for Running Database Designer

Vertica provides three ways to run Database Designer:

- [Creating a Database Design in Management Console](#)
- [Using Administration Tools to Create a Design](#)
- [Running Database Designer Programmatically](#)

The following workflow is common to all these ways to run Database Designer:



Logging Projection Data for Database Designer

When you run Database Designer, the Optimizer proposes a set of ideal projections based on the options that you specify. When you deploy the design, Database Designer creates the design based on these projections. However, space or budget constraints may prevent Database Designer from creating all the proposed projections. In addition, Database Designer may not be able to implement the projections using ideal criteria.

To get information about the projections, first enable the Database Designer logging capability. When enabled, Database Designer stores information about the proposed projections in two Data Collector tables. After Database Designer deploys the design, these logs contain information about which proposed projections were actually created. After deployment, the logs contain information about:

- Projections that the Optimizer proposed
- Projections that Database Designer actually created when the design was deployed
- Projections that Database Designer created, but not with the ideal criteria that the Optimizer identified.
- The DDL used to create all the projections
- Column optimizations

If you do not deploy the design immediately, review the log to determine if you want to make any changes. If the design has been deployed, you can still manually create some of the projections that Database Designer did not create.

To enable the Database Designer logging capability, see [Enabling Logging for Database Designer](#)

To view the logged information, see [Viewing Database Designer Logs](#).

Enabling Logging for Database Designer

By default, Database Designer does not log information about the projections that the Optimizer proposed and the Database Designer deploys.

To enable Database Designer logging, enter the following command:

```
=> ALTER DATABASE mydb SET DBDLogInternalDesignProcess = 1;
```

To disable Database Designer logging, enter the following command:

```
=> ALTER DATABASE mydb SET DBDLogInternalDesignProcess = 0;
```

For more information about logging, see:

- [Logging Projection Data for Database Designer](#)
- [Viewing Database Designer Logs](#)

Viewing Database Designer Logs

You can find data about the projections that Database Designer considered and deployed in two Data Collector tables:

- DC_DESIGN_PROJECTION_CANDIDATES
- DC_DESIGN_QUERY_PROJECTION_CANDIDATES

DC_DESIGN_PROJECTION_CANDIDATES

The DC_DESIGN_PROJECTION_CANDIDATES table contains information about all the projections that the Optimizer proposed. This table also includes the DDL that creates them. The `is_a_winner` field indicates if that projection was part of the actual deployed design. To view the DC_DESIGN_PROJECTION_CANDIDATES table, enter:

```
=> SELECT * FROM DC_DESIGN_PROJECTION_CANDIDATES;
```

DC_DESIGN_QUERY_PROJECTION_CANDIDATES

The DC_DESIGN_QUERY_PROJECTION_CANDIDATES table lists plan features for all design queries.

Possible features are:

- FULLY DISTRIBUTED JOIN
- MERGE JOIN
- GROUPBY PIPE
- FULLY DISTRIBUTED GROUPBY
- RLE PREDICATE

- VALUE INDEX PREDICATE
- LATE MATERIALIZATION

For all design queries, the DC_DESIGN_QUERY_PROJECTION_CANDIDATES table includes the following plan feature information:

- Optimizer path cost.
- Database Designer benefits.
- Ideal plan feature and its description, which identifies how the referenced projection should be optimized.
- If the design was deployed, the actual plan feature and its description is included in the table. This information identifies how the referenced projection was actually optimized.

Because most projections have multiple optimizations, each projection usually has multiple rows. To view the DC_DESIGN_QUERY_PROJECTION_CANDIDATES table, enter:

```
=> SELECT * FROM DC_DESIGN_QUERY_PROJECTION_CANDIDATES;
```

To see example data from these tables, see [Database Designer Logs: Example Data](#).

Database Designer Logs: Example Data

In the following example, Database Designer created the logs after creating a comprehensive design for the VMart sample database. The output shows two records from the DC_DESIGN_QUERY_PROJECTION_CANDIDATES table.

The first record contains information about the customer_dimension_dbd_1_sort_\$customer_gender__\$annual_income\$ projection. The record includes the CREATE PROJECTION statement that Database Designer used to create the projection. The is_a_winner column is t, indicating that Database Designer created this projection when it deployed the design.

The second record contains information about the product_dimension_dbd_2_sort_\$product_version__\$product_key\$ projection. For this projection, the is_a_winner column is f. The Optimizer recommended that Database Designer create this projection as part of the design. However, Database Designer did not create the projection when it deployed the design. The log includes the DDL for the CREATE PROJECTION statement. If you want to add the projection manually, you can use that DDL. For more information, see [Creating a Design Manually](#).

```
=> SELECT * FROM dc_design_projection_candidates;
-[ RECORD 1 ]-----+-----
time                | 2014-04-11 06:30:17.918764-07
node_name           | v_vmart_node0001
session_id          | localhost.localdoma-931:0x1b7
user_id             | 45035996273704962
user_name           | dbadmin
design_id            | 45035996273705182
design_table_id      | 45035996273720620
projection_id        | 45035996273726626
iteration_number     | 1
projection_name      | customer_dimension_dbd_1_sort_$customer_gender__$annual_income$
projection_statement | CREATE PROJECTION v_dbd_sarahtest_sarahtest."customer_dimension_dbd_1_
                    sort_$customer_gender__$annual_income$"
(
customer_key ENCODING AUTO,
customer_type ENCODING AUTO,
customer_name ENCODING AUTO,
customer_gender ENCODING RLE,
title ENCODING AUTO,
household_id ENCODING AUTO,
customer_address ENCODING AUTO,
customer_city ENCODING AUTO,
customer_state ENCODING AUTO,
customer_region ENCODING AUTO,
marital_status ENCODING AUTO,
customer_age ENCODING AUTO,
number_of_children ENCODING AUTO,
annual_income ENCODING AUTO,
occupation ENCODING AUTO,
largest_bill_amount ENCODING AUTO,
store_membership_card ENCODING AUTO,
customer_since ENCODING AUTO,
deal_stage ENCODING AUTO,
deal_size ENCODING AUTO,
last_deal_update ENCODING AUTO
)
AS
SELECT customer_key,
customer_type,
customer_name,
customer_gender,
title,
household_id,
customer_address,
customer_city,
customer_state,
customer_region,
marital_status,
customer_age,
number_of_children,
annual_income,
occupation,
largest_bill_amount,
store_membership_card,
customer_since,
deal_stage,
deal_size,
last_deal_update
FROM public.customer_dimension
```

```
ORDER BY customer_gender,
annual_income
UNSEGMENTED ALL NODES;
is_a_winner      | t
-[ RECORD 2 ]-----+-----
time             | 2014-04-11 06:30:17.961324-07
node_name        | v_vmart_node0001
session_id       | localhost.localdoma-931:0x1b7
user_id          | 45035996273704962
user_name        | dbadmin
design_id         | 45035996273705182
design_table_id  | 45035996273720624
projection_id    | 45035996273726714
iteration_number  | 1
projection_name  | product_dimension_dbd_2_sort_$product_version__$product_key$
projection_statement | CREATE PROJECTION v_dbd_sarahtest_sarahtest."product_dimension_dbd_2_
                sort_$product_version__$product_key$"
(
product_key ENCODING AUTO,
product_version ENCODING RLE,
product_description ENCODING AUTO,
sku_number ENCODING AUTO,
category_description ENCODING AUTO,
department_description ENCODING AUTO,
package_type_description ENCODING AUTO,
package_size ENCODING AUTO,
fat_content ENCODING AUTO,
diet_type ENCODING AUTO,
weight ENCODING AUTO,
weight_units_of_measure ENCODING AUTO,
shelf_width ENCODING AUTO,
shelf_height ENCODING AUTO,
shelf_depth ENCODING AUTO,
product_price ENCODING AUTO,
product_cost ENCODING AUTO,
lowest_competitor_price ENCODING AUTO,
highest_competitor_price ENCODING AUTO,
average_competitor_price ENCODING AUTO,
discontinued_flag ENCODING AUTO
)
AS
SELECT product_key,
product_version,
product_description,
sku_number,
category_description,
department_description,
package_type_description,
package_size,
fat_content,
diet_type,
weight,
weight_units_of_measure,
shelf_width,
shelf_height,
shelf_depth,
product_price,
product_cost,
lowest_competitor_price,
highest_competitor_price,
```

```

average_competitor_price,
discontinued_flag
FROM public.product_dimension
ORDER BY product_version,
product_key
UNSEGMENTED ALL NODES;
is_a_winner          | f
.
.
.

```

The next example shows the contents of two records in the DC_DESIGN_QUERY_PROJECTION_CANDIDATES. Both of these rows apply to projection id 45035996273726626.

In the first record, the Optimizer recommends that Database Designer optimize the customer_gender column for the GROUPBY PIPE algorithm.

In the second record, the Optimizer recommends that Database Designer optimize the public.customer_dimension table for late materialization. Late materialization can improve the performance of joins that might spill to disk.

```

=> SELECT * FROM dc_design_query_projection_candidates;
-[ RECORD 1 ]-----+-----
time                | 2014-04-11 06:30:17.482377-07
node_name           | v_vmart_node0001
session_id          | localhost.localdoma-931:0x1b7
user_id             | 45035996273704962
user_name           | dbadmin
design_id            | 45035996273705182
design_query_id      | 3
iteration_number     | 1
design_table_id      | 45035996273720620
projection_id        | 45035996273726626
ideal_plan_feature   | GROUP BY PIPE
ideal_plan_feature_description | Group-by pipelined on column(s) customer_gender
dbd_benefits        | 5
opt_path_cost       | 211
-[ RECORD 2 ]-----+-----
time                | 2014-04-11 06:30:17.48276-07
node_name           | v_vmart_node0001
session_id          | localhost.localdoma-931:0x1b7
user_id             | 45035996273704962
user_name           | dbadmin
design_id            | 45035996273705182
design_query_id      | 3
iteration_number     | 1
design_table_id      | 45035996273720620
projection_id        | 45035996273726626
ideal_plan_feature   | LATE MATERIALIZATION
ideal_plan_feature_description | Late materialization on table public.customer_dimension
dbd_benefits        | 4
opt_path_cost       | 669
.
.
.

```

You can view the actual plan features that Database Designer implemented for the projections it created. To do so, query the V_INTERNAL.DC_DESIGN_QUERY_PROJECTIONS table:

```
=> select * from v_internal.dc_design_query_projections;
-[ RECORD 1 ]-----+-----
time                | 2014-04-11 06:31:41.19199-07
node_name           | v_vmart_node0001
session_id          | localhost.localdoma-931:0x1b7
user_id             | 45035996273704962
user_name           | dbadmin
design_id            | 45035996273705182
design_query_id      | 1
projection_id       | 2
design_table_id      | 45035996273720624
actual_plan_feature | RLE PREDICATE
actual_plan_feature_description | RLE on predicate column(s) department_description
dbd_benefits        | 2
opt_path_cost       | 141
-[ RECORD 2 ]-----+-----
time                | 2014-04-11 06:31:41.192292-07
node_name           | v_vmart_node0001
session_id          | localhost.localdoma-931:0x1b7
user_id             | 45035996273704962
user_name           | dbadmin
design_id            | 45035996273705182
design_query_id      | 1
projection_id       | 2
design_table_id      | 45035996273720624
actual_plan_feature | GROUP BY PIPE
actual_plan_feature_description | Group-by pipelined on column(s) fat_content
dbd_benefits        | 5
opt_path_cost       | 155
```

Specifying Parameters for Database Designer

Before you run Database Designer to create a design, provide information that allows Database Designer to create the optimal physical schema:

- [Design Name](#)
- [Design Types](#)
- [Optimization Objectives](#)
- [Design Tables with Sample Data](#)
- [Design Queries](#)
- [K-safety](#)

- [Replicated and Segmented Projections](#)
- [Statistics Analysis](#)

Design Name

All designs that Database Designer creates must have a name that you specify. The design name must be alphanumeric or underscore (_) characters, and can be no more than 32 characters long. (Administrative Tools and Management Console limit the design name to 16 characters.)

The design name becomes part of the files that Database Designer generates, including the deployment script, allowing the files to be easily associated with a particular Database Designer run.

Design Types

The Database Designer can create two distinct design types. The design you choose depends on what you are trying to accomplish:

- [Comprehensive Design](#)
- [Incremental Design](#)

Comprehensive Design

A comprehensive design creates an initial or replacement design for all the tables in the specified schemas. Create a comprehensive design when you are creating a new database.

To help Database Designer create an efficient design, load representative data into the tables before you begin the design process. When you load data into a table, Vertica creates an unoptimized superprojection so that Database Designer has projections to optimize. If a table has no data, Database Designer cannot optimize it.

Optionally, supply Database Designer with representative queries that you plan to use so Database Designer can optimize the design for them. If you do not supply any queries, Database Designer creates a generic optimization of the superprojections that minimizes storage, with no query-specific projections.

During a comprehensive design, Database Designer creates deployment scripts that:

- Create new projections to optimize query performance, only when they do not already exist.
- Create replacement buddy projections when Database Designer changes the encoding of pre-existing projections that it has decided to keep.

Incremental Design

After you create and deploy a comprehensive database design, it's likely that your database will change over time in various ways. You should periodically consider using Database Designer to create incremental designs that address these changes. Changes that warrant an incremental design can include:

- Significant data additions or updates
- New or modified queries that you run regularly
- Performance issues with one or more queries
- Schema changes

Optimization Objectives

When creating a design, Database Designer can optimize the design for one of three objectives:

- **Load:** Database Designer creates a design that is optimized for loads, minimizing database size, potentially at the expense of query performance.
- **Performance:** Database Designer creates a design that is optimized for fast query performance. Because it recommends a design for optimized query performance, this design might recommend more than the Load or Balanced objectives, potentially resulting in a larger database storage size.
- **Balanced:** Database Designer creates a design whose objectives are balanced between database size and query performance.

A fully optimized query has an optimization ratio of 0.99. Optimization ratio is the ratio of a query's benefits achieved in the design produced by the Database Designer to that achieved in the ideal plan. Check the optimization ratio with the OptRatio parameter in designer.log.

Design Tables with Sample Data

You *must* specify one or more design tables for Database Designer to deploy a design. If your schema is empty, it does not appear as a design table option.

When you specify design tables, consider the following:

- To create the most efficient projections for your database, load a moderate amount of representative data into tables before running Database Designer. Database Designer considers the data in this table when creating the design.
- If your design tables have a large amount of data, the Database Designer run takes a long time; if your tables have too little data, the design is not optimized. Vertica recommends that 10 GB of sample data is sufficient for creating an optimal design.
- If you submit a design table with no data, Database Designer ignores it.
- If one of your design tables has been dropped, you will not be able to build or deploy your design.

Design Queries

If you supply representative queries that you run on your database to Database Designer, it optimizes the performance of those queries.

Database Designer checks the validity of all queries when you add them to your design and again when it builds the design. If a query is invalid, Database Designer ignores it.

The query file can contain up to 100 queries. Each query can be assigned a weight that indicates its relative importance so that Database Designer can prioritize it when creating the design. Database Designer groups queries that affect the design that Database Designer creates in the same way and considers one weighted query when creating a design.

The following options apply, depending on whether you create an incremental or comprehensive design:

- Design queries are required for incremental designs.
- Design queries are optional for comprehensive designs. If you do not provide design queries, Database Designer recommends a generic design that does not consider specific queries.

Query Repository

Using Management Console, you can submit design queries from the [QUERY_REQUESTS](#) system table. This is called *the query repository*.

The QUERY_REQUESTS table contains queries that users have run recently. For a comprehensive design, you can submit up to 200 queries from the QUERY_REQUESTS table to Database Designer to be considered when creating the design. For an incremental design, you can submit up to 100 queries from the QUERY_REQUESTS table.

Replicated and Segmented Projections

When creating a comprehensive design, Database Designer creates projections based on data statistics and queries. It also reviews the submitted design tables to decide whether projections should be segmented (distributed across the cluster nodes) or replicated (duplicated on all cluster nodes).

For detailed information, see the following sections:

- [Replicated Projections](#)
- [Segmented Projections](#)

Replicated Projections

Replication occurs when Vertica stores identical copies of data across all nodes in a cluster.

If you are running on a single-node database, *all* projections are replicated because segmentation is not possible in a single-node database.

Assuming that *largest-row-count* equals the number of rows in the design table with the largest number of rows, Database Designer recommends that a projection be replicated if any of the following conditions is true:

- *largest-row-count* < 1,000,000 and number of rows in the table <= 10% of *largest-row-count*
- *largest-row-count* >= 10,000,000 and number of rows in the table <= 1% of *largest-row-count*
- The number of rows in the table <= 100,000

For more information about replication, see [High Availability With Projections](#) in Vertica Concepts.

Segmented Projections

Segmentation occurs when Vertica distributes data evenly across multiple database nodes so that all nodes participate in query execution. Projection segmentation provides high availability and recovery, and optimizes query execution.

When running Database Designer programmatically or using Management Console, you can specify to allow Database Designer to recommend unsegmented projections in the design. If you do not specify this, Database Designer recommends only segmented projections.

Database Designer recommends segmented superprojections for large tables when deploying to multiple node clusters, and recommends replicated superprojections for smaller tables.

Database Designer does not segment projections on:

- Single-node clusters
- LONG VARCHAR and LONG VARBINARY columns

For more information about segmentation, see [High Availability With Projections](#) in Vertica Concepts.

Statistics Analysis

By default, Database Designer analyzes statistics for the design tables when adding them to the design. This option is optional, but Vertica recommends that you analyze statistics because accurate statistics help Database Designer optimize compression and query performance.

Analyzing statistics takes time and resources. If the current statistics for the design tables are up to date, do not bother analyzing the statistics. When in doubt, analyze the statistics to make sure they are current.

For more information, see [Collecting Statistics](#).

Building a Design

After you have created design tables and loaded data into them, and then specified the parameters you want Database Designer to use when creating the physical schema, direct Database Designer to create the scripts necessary to build the design.

Note: You cannot stop a running database if Database Designer is building a database design.

When you build a database design, Vertica generates two scripts:

- **Deployment script:** *design-name_deploy.sql*—Contains the SQL statements that create projections for the design you are deploying, deploy the design, and drop unused projections. When the deployment script runs, it creates the optimized design. For details about how to run this script and deploy the design, see [Deploying a Design](#).
- **Design script:** *design-name_design.sql*—Contains the CREATE PROJECTION statements that Database Designer uses to create the design. Review this script to make sure you are happy with the design.

The design script is a subset of the deployment script. It serves as a backup of the DDL for the projections that the deployment script creates.

When you create a design using Management Console:

- If you submit a large number of queries to your design and build it right immediately, a timing issue could cause the queries not to load before deployment starts. If this occurs, you might see one of the following errors:
 - No queries to optimize for
 - No tables to design projections for

To accommodate this timing issue, you may need to reset the design, check the **Queries** tab to make sure the queries have been loaded, and then rebuild the design. Detailed instructions are in:

- [Using the Wizard to Create a Design](#)
- [Creating a Design Manually](#)
- The scripts are deleted when deployment completes. To save a copy of the deployment script after the design is built but before the deployment completes, go to the **Output** window and copy and paste the SQL statements to a file.

Resetting a Design

You must reset a design when:

- You build a design and the output scripts described in [Building a Design](#) are not created.
- You build a design but Database Designer cannot complete the design because the queries it expects are not loaded.

Resetting a design discards all the run-specific information of the previous Database Designer build, but retains its configuration (design type, optimization objectives, K-safety, etc.) and tables and queries.

After you reset a design, review the design to see what changes you need to make. For example, you can fix errors, change parameters, or check for and add additional tables or queries. Then you can rebuild the design.

You can only reset a design in Management Console or by using the [DESIGNER_RESET_DESIGN](#) function.

Deploying a Design

After running Database Designer to generate a deployment script, Vertica recommends that you test your design on a non-production server before you deploy it to your production server.

Both the design and deployment processes run in the background. This is useful if you have a large design that you want to run overnight. Because an active SSH session is not required, the design/deploy operations continue to run uninterrupted, even if the session is terminated.

Note: You cannot stop a running database if Database Designer is building or deploying a database design.

Database Designer runs as a background process. Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. However, if multiple users are deploying a design on the same tables at the same time, Database Designer may not be able to complete the deployment. To avoid problems, consider the following:

- Schedule potentially conflicting Database Designer processes to run sequentially overnight so that there are no concurrency problems.
- Avoid scheduling Database Designer runs on the same set of tables at the same time.

There are two ways to deploy your design:

- [Deploying Designs Using Database Designer](#)
- [Deploying Designs Manually](#)

Deploying Designs Using Database Designer

Micro Focus recommends that you run Database Designer and deploy optimized projections right after loading your tables with sample data because Database Designer provides projections optimized for the current state of your database.

If you choose to allow Database Designer to automatically deploy your script during a comprehensive design and are running Administrative Tools, Database Designer creates a backup script of your database's current design. This script helps you re-create the design of projections that may have been dropped by the new design. The backup script is located in the output directory you specified during the design process.

If you choose not to have Database Designer automatically run the deployment script (for example, if you want to maintain projections from a pre-existing deployment), you can manually run the deployment script later. See [Deploying Designs Manually](#).

To deploy a design while running Database Designer, do one of the following:

- In Management Console, select the design and click **Deploy Design**.
- In the Administration Tools, select **Deploy design** in the **Design Options** window.

If you are running Database Designer programmatically, use `DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY` and set the `deploy` parameter to 'true'.

Once you have deployed your design, query the `DEPLOY_STATUS` system table to see the steps that the deployment took:

```
vmartdb=> SELECT * FROM V_MONITOR.DEPLOY_STATUS;
```

Deploying Designs Manually

If you choose not to have Database Designer deploy your design at design time, you can deploy the design later using the deployment script:

1. Make sure that you have a database that contains the same tables and projections as the database on which you ran Database Designer. The database should also contain sample data.
2. To deploy the projections to a test or production environment, use the following `vsq` command to execute the deployment script, where *design-name* is the name of the database design:

```
=> \i design-name_deploy.sql
```

How to Create a Design

There are three ways to create a design using Database Designer:

- From Management Console, open a database and select the **Design** page at the bottom of the window.

For details about using Management Console to create a design, see [Creating a Database Design in Management Console](#)

- Programmatically, using the techniques described in [About Running Database Designer Programmatically](#) in Analyzing Data. To run Database Designer programmatically, you must be a **DBADMIN** or have been granted the **DBDUSER** role and enabled that role.
- From the Administration Tools menu, by selecting **Configuration Menu > Run Database Designer**. You must be a DBADMIN user to run Database Designer from the Administration Tools.

For details about using Administration Tools to create a design, see [Using Administration Tools to Create a Design](#).

The following table shows what Database Designer capabilities are available in each tool:

Database Designer Capability	Management Console	Running Database Designer Programmatically	Administrative Tools
Create design	Yes	Yes	Yes
Design name length (# of characters)	16	32	16
Build design (create design and deployment scripts)	Yes	Yes	Yes
Create backup script			Yes
Set design type (comprehensive or incremental)	Yes	Yes	Yes
Set optimization objective	Yes	Yes	Yes
Add design tables	Yes	Yes	Yes
Add design queries file	Yes	Yes	Yes
Add single design query		Yes	

Database Designer Capability	Management Console	Running Database Designer Programmatically	Administrative Tools
Use query repository	Yes	Yes	
Set K-safety	Yes	Yes	Yes
Analyze statistics	Yes	Yes	Yes
Require all unsegmented projections	Yes	Yes	
View event history	Yes	Yes	
Set correlation analysis mode (Default = 0)		Yes	

Using Administration Tools to Create a Design

To use the Administration Tools interface to create an optimized design for your database, you must be a DBADMIN user. Follow these steps:

1. Log in as the dbadmin user and start Administration Tools.
2. From the main menu, start the database for which you want to create a design. The database must be running before you can create a design for it.
3. On the main menu, select **Configuration Menu** and click **OK**.
4. On the Configuration Menu, select **Run Database Designer** and click **OK**.
5. On the **Select a database to design** window, enter the name of the database for which you are creating a design and click **OK**.
6. On the **Enter the directory for Database Designer output** window, enter the full path to the directory to contain the design script, deployment script, backup script, and log files, and click **OK**.

For information about the scripts, see [Building a Design](#).

7. On the **Database Designer** window, enter a name for the design and click **OK**.

For more information about design names, see [Design Name](#).

8. On the **Design Type** window, choose which type of design to create and click **OK**.

For a description of the design types, see [Design Types](#)

9. The **Select schema(s) to add to query search path** window lists all the schemas in the database that you selected. Select the schemas that contain representative data that you want Database Designer to consider when creating the design and click **OK**.

For more information about choosing schema and tables to submit to Database Designer, see [Design Tables with Sample Data](#).

10. On the **Optimization Objectives** window, select the objective you want for the database optimization:

- **Optimize with Queries**

For more information, see [Design Queries](#).

- **Update statistics**

For more information see [Statistics Analysis](#).

- **Deploy design**

For more information, see [Deploying a Design](#).

For details about these objectives, see [Optimization Objectives](#).

11. The final window summarizes the choices you have made and offers you two choices:

- **Proceed** with building the design, and deploying it if you specified to deploy it immediately. If you did not specify to deploy, you can review the design and deployment scripts and deploy them manually, as described in [Deploying Designs Manually](#).

- **Cancel** the design and go back to change some of the parameters as needed.

12. Creating a design can take a long time. To cancel a running design from the Administration Tools window, enter **Ctrl+C**.

To create a design for the VMart example database, see [Using Database Designer to Create a Comprehensive Design](#) in Getting Started.

Running Database Designer Programmatically

Vertica provides a set of meta-functions that enable programmatic access to Database Designer functionality. Run Database Designer programmatically to perform the following tasks:

- Optimize performance on tables that you own.
- Create or update a design without requiring superuser or DBADMIN intervention.
- Add individual queries and tables, or add data to your design, and then rerun Database Designer to update the design based on this new information.
- Customize the design.
- Use recently executed queries to set up your database to run Database Designer automatically on a regular basis.
- Assign each design query a *query weight* that indicates the importance of that query in creating the design. Assign a higher weight to queries that you run frequently so that Database Designer prioritizes those queries in creating the design.

For more details about Database Designer functions, see [Database Designer Function Categories](#).

Database Designer Function Categories

Database Designer functions perform the following operations, generally performed in the following order:

1. [Create a design](#).
2. [Set design properties](#).
3. [Populate a design](#).
4. [Create design and deployment scripts](#).
5. [Get design data](#).
6. [Clean up](#).

For detailed information, see [Workflow for Running Database Designer Programmatically](#).

Create a design

[DESIGNER_CREATE_DESIGN](#) directs Database Designer to create a design.

Set design properties

The following functions let you specify properties of a particular design:

DESIGNER_SET_DESIGN_TYPE	Specifies whether the design is comprehensive or incremental.
DESIGNER_DESIGN_PROJECTION_ENCODINGS	Analyzes encoding in the specified projections and creates a script that implements encoding recommendations.
DESIGNER_SET_DESIGN_KSAFETY	Sets the K-safety value for a comprehensive design.
DESIGNER_SET_OPTIMIZATION_OBJECTIVE	Specifies whether the design optimizes for query or load performance.
DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS	Enables inclusion of unsegmented projections in the design.
DESIGNER_SET_ANALYZE_CORRELATIONS_MODE	Determines how the design handles column correlations.

Populate a design

The following functions let you add tables and queries to your Database Designer design:

DESIGNER_ADD_DESIGN_TABLES	Adds the specified tables to a design.
DESIGNER_ADD_DESIGN_QUERY	Adds queries to the design and weights them.
DESIGNER_ADD_DESIGN_QUERIES	
DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS	

Create design and deployment scripts

The following functions populate the Database Designer workspace and create design and deployment scripts. You can also analyze statistics, deploy the design automatically, and drop the workspace after the deployment:

DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY	Populates the design and creates design and deployment scripts.
DESIGNER_WAIT_FOR_DESIGN	Waits for a currently running design to complete.

Reset a design

[DESIGNER_RESET_DESIGN](#) discards all the run-specific information of the previous Database Designer build or deployment of the specified design but retains its configuration.

Get design data

The following functions display information about projections and scripts that the Database Designer created:

DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS	Sends to standard output DDL statements that define design projections.
DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT	Sends to standard output a design's deployment script.

Clean up

The following functions cancel any running Database Designer operation or drop a Database Designer design and all its contents:

DESIGNER_CANCEL_POPULATE_DESIGN	Cancels population or deployment operation for the specified design if it is currently running.
DESIGNER_DROP_DESIGN	Removes the schema associated with the specified design and all its contents.
DESIGNER_DROP_ALL_DESIGNS	Removes all Database Designer-related schemas associated with the current user.

Workflow for Running Database Designer Programmatically

The following example shows the steps you take to create a design by running Database Designer programmatically.

Note: Be sure to back up the existing design using the [EXPORT_CATALOG](#) function before running the Database Designer functions on an existing schema. You must explicitly back up the current design when using Database Designer to create a new comprehensive design.

Before you run this example, you should have the DBDUSER role, and you should have enabled that role using the SET ROLE DBDUSER command:

1. Create a table in the public schema:

```
=> CREATE TABLE T(  
  x INT,  
  y INT,  
  z INT,  
  u INT,  
  v INT,  
  w INT PRIMARY KEY  
);
```

2. Add data to the table:

```
\! perl -e 'for ($i=0; $i<100000; ++$i) {printf("%d, %d, %d, %d, %d, %d\n", $i/10000, $i/100,  
$i/10, $i/2, $i, $i);}'  
| vsql -c "COPY T FROM STDIN DELIMITER ',' DIRECT;"
```

3. Create a second table in the public schema:

```
=> CREATE TABLE T2(  
  x INT,  
  y INT,  
  z INT,  
  u INT,  
  v INT,  
  w INT PRIMARY KEY  
);
```

4. Copy the data from table T1 to table T2 and commit the changes:

```
=> INSERT /*+DIRECT*/ INTO T2 SELECT * FROM T;  
=> COMMIT;
```

5. Create a new design:

```
=> SELECT DESIGNER_CREATE_DESIGN('my_design');
```

This command adds information to the [DESIGNS](#) system table in the V_MONITOR schema.

6. Add tables from the public schema to the design :

```
=> SELECT DESIGNER_ADD_DESIGN_TABLES('my_design', 'public.t');  
=> SELECT DESIGNER_ADD_DESIGN_TABLES('my_design', 'public.t2');
```

These commands add information to the [DESIGN_TABLES](#) system table.

7. Create a file named `queries.txt` in `/tmp/examples`, or another directory where you have READ and WRITE privileges. Add the following two queries in that file and save it. Database Designer uses these queries to create the design:

```
SELECT DISTINCT T2.u FROM T JOIN T2 ON T.z=T2.z-1 WHERE T2.u > 0;  
SELECT DISTINCT w FROM T;
```

8. Add the queries file to the design and display the results—the numbers of accepted queries, non-design queries, and unoptimizable queries:

```
=> SELECT DESIGNER_ADD_DESIGN_QUERIES  
      ('my_design',  
       '/tmp/examples/queries.txt',  
       'true'  
      );
```

The results show that both queries were accepted:

```
Number of accepted queries           =2  
Number of queries referencing non-design tables =0  
Number of unsupported queries       =0  
Number of illegal queries           =0
```

The `DESIGNER_ADD_DESIGN_QUERIES` function populates the [DESIGN_QUERIES](#) system table.

9. Set the design type to **comprehensive**. (This is the default.) A comprehensive design creates an initial or replacement design for all the design tables:

```
=> SELECT DESIGNER_SET_DESIGN_TYPE('my_design', 'comprehensive');
```

10. Set the optimization objective to **query**. This setting creates a design that focuses on faster query performance, which might recommend additional projections. These projections could result in a larger database storage footprint:

```
=> SELECT DESIGNER_SET_OPTIMIZATION_OBJECTIVE('my_design', 'query');
```

11. Create the design and save the design and deployment scripts in `/tmp/examples`, or another directory where you have READ and WRITE privileges. The following command:

- Analyzes statistics
- Doesn't deploy the design.
- Doesn't drop the design after deployment.
- Stops if it encounters an error.

```
=> SELECT DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY  
  ('my_design',  
   '/tmp/examples/my_design_projections.sql',  
   '/tmp/examples/my_design_deploy.sql',  
   'True',  
   'False',  
   'False',  
   'False'  
  );
```

This command adds information to the following system tables:

- [DEPLOYMENT_PROJECTION_STATEMENTS](#)
- [DEPLOYMENT_PROJECTIONS](#)
- [OUTPUT_DEPLOYMENT_STATUS](#)

12. Examine the status of the Database Designer run to see what projections Database Designer recommends. In the `deployment_projection_name` column:

- `rep` indicates a replicated projection
- `super` indicates a superprojection

The `deployment_status` column is pending because the design has not yet been deployed.

For this example, Database Designer recommends four projections:

```
=> \x
Expanded display is on.
=> SELECT * FROM OUTPUT_DEPLOYMENT_STATUS;
-[ RECORD 1 ]-----+-----
deployment_id      | 45035996273795970
deployment_projection_id | 1
deployment_projection_name | T_DBD_1_rep_my_design
deployment_status   | pending
error_message      | N/A
-[ RECORD 2 ]-----+-----
deployment_id      | 45035996273795970
deployment_projection_id | 2
deployment_projection_name | T2_DBD_2_rep_my_design
deployment_status   | pending
error_message      | N/A
-[ RECORD 3 ]-----+-----
deployment_id      | 45035996273795970
deployment_projection_id | 3
deployment_projection_name | T_super
deployment_status   | pending
error_message      | N/A
-[ RECORD 4 ]-----+-----
deployment_id      | 45035996273795970
deployment_projection_id | 4
deployment_projection_name | T2_super
deployment_status   | pending
error_message      | N/A
```

13. View the script `/tmp/examples/my_design_deploy.sql` to see how these projections are created when you run the deployment script. In this example, the script also assigns the encoding schemes `RLE` and `COMMONDELTA_COMP` to columns where appropriate.
14. Deploy the design from the directory where you saved it:

```
=> \i /tmp/examples/my_design_deploy.sql
```

15. Now that the design is deployed, delete the design:

```
=> SELECT DESIGNER_DROP_DESIGN('my_design');
```

Privileges for Running Database Designer Functions

Non-DBADMIN users with the [DBDUSER role](#) can run Database Designer [functions](#). Two steps are required to enable users to run these functions:

1. A DBADMIN or superuser grants the user the DBDUSER role:

```
=> GRANT DBDUSER TO username;
```

This role persists until the DBADMIN revokes it.

2. Before the DBDUSER can run Database Designer functions, one of the following must occur:

- The user enables the DBDUSER role:

```
=> SET ROLE DBDUSER;
```

- The superuser sets the user's default role to DBDUSER:

```
=> ALTER USER username DEFAULT ROLE DBDUSER;
```

General DBDUSER Limitations

As a DBDUSER, the following restrictions apply:

- You can set a design's [K-safety](#) to a value less than or equal to system K-safety. You cannot change system K-safety.
- You cannot explicitly change the ancient history mark (AHM), even during design deployment.

Design Dependencies and Privileges

Individual design tasks are likely to have dependencies that require specific privileges:

Task	Required privileges
Add tables to a design	<ul style="list-style-type: none">• USAGE privilege on the design table schema• OWNER privilege on the design table
Add a single design query to the design	<ul style="list-style-type: none">• Privilege to execute the design query
Add a query file to the design	<ul style="list-style-type: none">• Read privilege on the storage location that contains the query file

Task	Required privileges
	<ul style="list-style-type: none">• Privilege to execute all the queries in the file
Add queries from the result of a user query to the design	<ul style="list-style-type: none">• Privilege to execute the user query• Privilege to execute each design query retrieved from the results of the user query
Create design and deployment scripts	<ul style="list-style-type: none">• WRITE privilege on the storage location of the design script• WRITE privilege on the storage location of the deployment script

Resource Pool for Database Designer Users

When you grant a user the DBDUSER role, be sure to associate a resource pool with that user to manage resources during Database Designer runs. This allows multiple users to run Database Designer concurrently without interfering with each other or using up all cluster resources.

Note: When a user runs Database Designer, execution is mostly contained in the user's resource pool. However, Vertica might also use other system resource pools to perform less-intensive tasks.

Creating Custom Designs

Micro Focus strongly recommends that you use the physical schema design produced by Database Designer, which provides K-safety, excellent query performance, and efficient use of storage space. If any queries run less as efficiently than you expect, consider using the Database Designer incremental design process to optimize the database design for the query.

If the projections created by Database Designer still do not meet your needs, you can write custom projections, from scratch or based on projection designs created by Database Designer.

If you are unfamiliar with writing custom projections, start by modifying an existing design generated by Database Designer.

Custom Design Process

To create a custom design or customize an existing one:

1. Plan the new design or modifications to an existing one. See [Planning Your Design](#).
2. Create or modify projections. See [Design Fundamentals](#) and [CREATE PROJECTION](#) for more detail.
3. Deploy projections to a test environment. See [Writing and Deploying Custom Projections](#).
4. Test and modify projections as needed.
5. After you finalize the design, deploy projections to the production environment.

Planning Your Design

The syntax for creating a design is easy for anyone who is familiar with SQL. As with any successful project, however, a successful design requires some initial planning. Before you create your first design:

- Become familiar with standard design requirements and plan your design to include them. See [Design Requirements](#).
- Determine how many projections you need to include in the design. See [Determining the Number of Projections to Use](#).
- Determine the type of compression and encoding to use for columns. See [Data Encoding and Compression](#).
- Determine whether or not you want the database to be K-safe. Vertica recommends that all production databases have a minimum K-safety of one (K=1). Valid K-safety values are 0, 1, and 2. See [Using Database Designer](#).

Design Requirements

A physical schema design is a script that contains CREATE PROJECTION statements. These statements determine which columns are included in projections and how they are optimized.

If you use Database Designer as a starting point, it automatically creates designs that meet all fundamental design requirements. If you intend to create or modify designs manually, be aware that all designs must meet the following requirements:

- Every design must create at least one superprojection for every table in the database that is used by the client application. These projections provide complete coverage that enables users to perform ad-hoc queries as needed. They can contain joins and they are usually configured to maximize performance through sort order, compression, and encoding.
- Query-specific projections are optional. If you are satisfied with the performance provided through superprojections, you do not need to create additional projections. However, you can maximize performance by tuning for specific query work loads.
- Micro Focus recommends that all production databases have a minimum K-safety of one (K=1) to support high availability and recovery. (K-safety can be set to 0, 1, or 2.) See [High Availability With Projections](#) in Vertica Concepts and [Using Database Designer](#).

- Vertica recommends that if you have more than 20 nodes, but small tables, do not create replicated projections. If you create replicated projections, the catalog becomes very large and performance may degrade. Instead, consider segmenting those projections.

Determining the Number of Projections to Use

In many cases, a design that consists of a set of superprojections (and their buddies) provides satisfactory performance through compression and encoding. This is especially true if the sort orders for the projections have been used to maximize performance for one or more query predicates (WHERE clauses).

However, you might want to add additional query-specific projections to increase the performance of queries that run slowly, are used frequently, or are run as part of business-critical reporting. The number of additional projections (and their buddies) that you create should be determined by:

- Your organization's needs
- The amount of disk space you have available on each node in the cluster
- The amount of time available for loading data into the database

As the number of projections that are tuned for specific queries increases, the performance of these queries improves. However, the amount of disk space used and the amount of time required to load data increases as well. Therefore, you should create and test designs to determine the optimum number of projections for your database configuration. On average, organizations that choose to implement query-specific projections achieve optimal performance through the addition of a few query-specific projections.

Designing for K-Safety

Micro Focus recommends that all production databases have a minimum K-safety of one ($K=1$). Valid K-safety values for production databases are 1 and 2. Non-production databases do not have to be K-safe and can be set to 0.

A K-safe database must have at least three nodes, as shown in the following table:

K-safety level	Number of required nodes
1	3+
2	5+

Note: Vertica only supports K-safety levels 1 and 2.

You can set K-safety to 1 or 2 only when the physical schema design meets certain redundancy requirements. See [Requirements for a K-Safe Physical Schema Design](#).

Using Database Designer

To create designs that are K-safe, Micro Focus recommends that you use the Database Designer. When creating projections with Database Designer, projection definitions that meet K-safe design requirements are recommended and marked with a K-safety level. Database Designer creates a script that uses the [MARK_DESIGN_KSAFE](#) function to set the K-safety of the physical schema to 1. For example:

```
=> \i VMart_Schema_design_opt_1.sql  
CREATE PROJECTION  
CREATE PROJECTION  
mark_design_ksafe  
-----  
Marked design 1-safe  
(1 row)
```

By default, Vertica creates K-safe superprojections when database K-safety is greater than 0.

Monitoring K-Safety

Monitoring tables can be accessed programmatically to enable external actions, such as alerts. You monitor the K-safety level by querying the [SYSTEM](#) table for settings in columns `DESIGNED_FAULT_TOLERANCE` and `CURRENT_FAULT_TOLERANCE`.

Loss of K-Safety

When K nodes in your cluster fail, your database continues to run, although performance is affected. Further node failures could potentially cause the database to shut down if the failed node's data is not available from another functioning node in the cluster.

See Also

[K-Safety](#) in Vertica Concepts

Requirements for a K-Safe Physical Schema Design

Database Designer automatically generates designs with a K-safety of 1 for clusters that contain at least three nodes. (If your cluster has one or two nodes, it generates designs with a K-safety of 0. You can modify a design created for a three-node (or greater) cluster, and the K-safe requirements are already set.

If you create custom projections, your physical schema design must meet the following requirements to be able to successfully recover the database in the event of a failure:

- Segmented projections must be segmented across all nodes. Refer to [Designing for Segmentation](#) and [Designing Segmented Projections for K-Safety](#).
- Replicated projections must be replicated on all nodes. See [Designing Unsegmented Projections for K-Safety](#).
- Segmented projections must have K+1 buddy projections—projections with identical columns and segmentation criteria, where corresponding segments are placed on different nodes.

You can use the [MARK_DESIGN_KSAFE](#) function to find out whether your schema design meets requirements for K-safety.

Requirements for a Physical Schema Design with No K-Safety

If you use Database Designer to generate an comprehensive design that you can modify and you do not want the design to be K-safe, set K-safety level to 0 (zero).

If you want to start from scratch, do the following to establish minimal projection requirements for a functioning database with no K-safety (K=0):

1. Define at least one superprojection for each table in the logical schema.
2. Replicate (define an exact copy of) each dimension table superprojection on each node.

Designing Segmented Projections for K-Safety

Projections must comply with database K-safety requirements. In general, you must create buddy projections for each segmented projection, where the number of buddy projections is $K+1$. Thus, if system K-safety is set to 1, each projection segment must be duplicated by one buddy; if K-safety is set to 2, each segment must be duplicated by two buddies.

Automatic Creation of Buddy Projections

You can use `CREATE PROJECTION` so it automatically creates the number of buddy projections required to satisfy K-safety, by including `SEGMENTED BY ... ALL NODES`. If `CREATE PROJECTION` specifies K-safety (`KSAFE=n`), Vertica uses that setting; if the statement omits `KSAFE`, Vertica uses system K-safety.

In the following example, `CREATE PROJECTION` creates segmented projection `ttt_p1` for table `ttt`. Because system K-safety is set to 1, Vertica requires a buddy projection for each segmented projection. The `CREATE PROJECTION` statement omits `KSAFE`, so Vertica uses system K-safety and creates two buddy projections: `ttt_p1_b0` and `ttt_p1_b1`:

```
=> SELECT mark_design_ksafe(1);

  mark_design_ksafe
-----
Marked design 1-safe
(1 row)

=> CREATE TABLE ttt (a int, b int);
WARNING 6978: Table "ttt" will include privileges from schema "public"
CREATE TABLE

=> CREATE PROJECTION ttt_p1 as SELECT * FROM ttt SEGMENTED BY HASH(a) ALL NODES;
CREATE PROJECTION

=> SELECT projection_name from projections WHERE anchor_table_name='ttt';
  projection_name
-----
  ttt_p1_b0
  ttt_p1_b1
(2 rows)
```

Vertica automatically names buddy projections by appending the suffix `_bn` to the projection base name—for example `ttt_p1_b0`.

Manual Creation of Buddy Projections

If you create a projection on a single node, and system K-safety is greater than 0, you must manually create the number of buddies required for K-safety. For example, you can create projection `xxx_p1` for table `xxx` on a single node, as follows:

```
=> CREATE TABLE xxx (a int, b int);  
WARNING 6978: Table "xxx" will include privileges from schema "public"  
CREATE TABLE  
  
=> CREATE PROJECTION xxx_p1 AS SELECT * FROM xxx SEGMENTED BY HASH(a) NODES v_vmart_node0001;  
CREATE PROJECTION
```

Because K-safety is set to 1, a single instance of this projection is not K-safe. Attempts to insert data into its anchor table `xxx` return with an error like this:

```
=> INSERT INTO xxx VALUES (1, 2);  
ERROR 3586: Insufficient projections to answer query  
DETAIL: No projections that satisfy K-safety found for table xxx  
HINT: Define buddy projections for table xxx
```

In order to comply with K-safety, you must create a buddy projection for projection `xxx_p1`. For example:

```
=> CREATE PROJECTION xxx_p1_buddy AS SELECT * FROM xxx SEGMENTED BY HASH(a) NODES v_vmart_node0002;  
CREATE PROJECTION
```

Table `xxx` now complies with K-safety and accepts DML statements such as `INSERT`:

```
VMart=> INSERT INTO xxx VALUES (1, 2);  
OUTPUT  
-----  
      1  
(1 row)
```

See Also

For general information about segmented projections and buddies, see [Projection Segmentation](#) in Vertica Concepts. For information about designing for K-safety, see [Using Database Designer](#) and [Designing for Segmentation](#).

Designing Unsegmented Projections for K-Safety

In many cases, dimension tables are relatively small, so you do not need to segment them. Accordingly, you should design a K-safe database so projections for its dimension tables are replicated without segmentation on all cluster nodes. You create these projections with a `CREATE PROJECTION` statement that includes the keywords `UNSEGMENTED ALL NODES`. These keywords specify to create identical instances of the projection on all cluster nodes.

The following example shows how to create an unsegmented projection for the table `store.store_dimension`:

```
=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state)
      AS SELECT store_key, store_name, store_city, store_state
      FROM store.store_dimension
      UNSEGMENTED ALL NODES;
CREATE PROJECTION
```

Vertica uses the same name to identify all instances of the unsegmented projection—in this example, `store.store_dimension_proj`. The keyword `ALL NODES` specifies to replicate the projection on all nodes:

```
=> \dj store.store_dimension_proj
      List of projections
Schema | Name | Owner | Node | Comment
-----+-----+-----+-----+-----
store | store_dimension_proj | dbadmin | v_vmart_node0001 |
store | store_dimension_proj | dbadmin | v_vmart_node0002 |
store | store_dimension_proj | dbadmin | v_vmart_node0003 |
(3 rows)
```

For more information about projection name conventions, see [Projection Naming](#).

Designing for Segmentation

You segment projections using hash segmentation. Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution. In a projection, the data to be hashed consists of one or more column values, each having a large number of unique values and an acceptable amount of skew in the value distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation.

Note: For detailed information about using hash segmentation in a projection, see [CREATE PROJECTION](#) in the SQL Reference Manual.

When segmenting projections, determine which columns to use to segment the projection. Choose one or more columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns are an excellent choice for hash segmentation. The columns must be unique across all the tables being used in a query.

Design Fundamentals

Although you can write custom projections from scratch, Vertica recommends that you use Database Designer to create a design to use as a starting point. This ensures that you have projections that meet basic requirements.

Writing and Deploying Custom Projections

Before you write custom projections, be sure to review the topics in [Planning Your Design](#) carefully. Failure to follow these considerations can result in non-functional projections.

To manually modify or create a projection:

1. Write a script to create the projection, using the [CREATE PROJECTION](#) statement.
2. Use the `\i` meta-command in vsql to run the script.

Note: You must have a database loaded with a logical schema.

3. For a K-safe database, use the function `SELECT get_projections('table_name')` to verify that the projections were properly created. Good projections are noted as being "safe." This means that the projection has enough buddies to be K-safe.
4. If you added the new projection to a database that already has projections that contain data, you need to update the newly created projection to work with the existing projections. By default, the new projection is out-of-date (not available for query processing) until you refresh it.
5. Use the [MAKE_AHM_NOW](#) function to set the Ancient History Mark (AHM) to the greatest allowable epoch (now).
6. Use [DROP PROJECTION](#) to drop any previous projections that are no longer needed.

These projections can waste disk space and reduce load speed if they remain in the database.

7. Run the [ANALYZE_STATISTICS](#) function on all projections in the database. This function collects and aggregates data samples and storage information from all nodes on which a projection is stored, and then writes statistics into the catalog. For example:

```
=>SELECT ANALYZE_STATISTICS ('');
```

Designing Superprojections

Superprojections have the following requirements:

- They must contain every column within the table.
- For a K-safe design, superprojections must either be replicated on all nodes within the database cluster (for dimension tables) or paired with buddies and segmented across all nodes (for very large tables and medium large tables). See [Physical Schema](#) and [High Availability With Projections](#) in Vertica Concepts for an overview of projections and how they are stored. See [Using Database Designer](#) for design specifics.

To provide maximum usability, superprojections need to minimize storage requirements while maximizing query performance. To achieve this, the sort order for columns in superprojections is based on storage requirements and commonly used queries.

Sort Order Benefits

Column sort order is an important factor in minimizing storage requirements, and maximizing query performance.

Minimize Storage Requirements

Minimizing storage saves on physical resources and increases performance by reducing disk I/O. You can minimize projection storage by prioritizing low-cardinality columns in its sort order. This reduces the number of rows Vertica stores and accesses to retrieve query results.

After identifying projection sort columns, analyze their data and choose the most effective encoding method. The Vertica optimizer gives preference to columns with run-length encoding (RLE), so be sure to use it whenever appropriate. Run-length encoding replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, it is especially appropriate to use it for low-cardinality columns whose run length is large.

Maximize Query Performance

You can facilitate query performance through column sort order as follows:

- Where possible, sort order should prioritize columns with the lowest cardinality.
- Do not sort projections on columns of type LONG VARBINARY and LONG VARCHAR.

For more information

See [Combine RLE and Sort Order](#) for examples that address storage and query requirements.

Choosing Sort Order: Best Practices

When choosing sort orders for your projections, Vertica has several recommendations that can help you achieve maximum query performance, as illustrated in the following examples.

Combine RLE and Sort Order

When dealing with predicates on low-cardinality columns, use a combination of RLE and sorting to minimize storage requirements and maximize query performance.

Suppose you have a `students` table contain the following values and encoding types:

Column	# of Distinct Values	Encoded With
<code>gender</code>	2 (M or F)	RLE
<code>pass_fail</code>	2 (P or F)	RLE
<code>class</code>	4 (freshman, sophomore, junior, or senior)	RLE
<code>name</code>	10000 (too many to list)	Auto

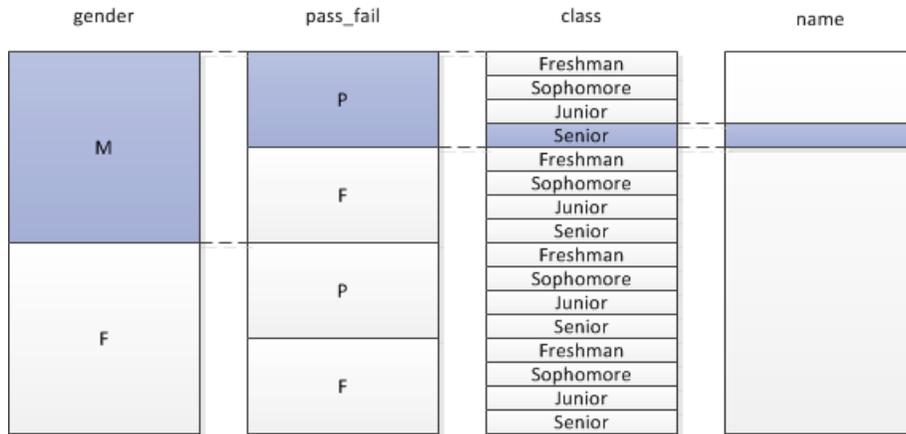
You might have queries similar to this one:

```
SELECT name FROM students WHERE gender = 'M' AND pass_fail = 'P' AND class = 'senior';
```

The fastest way to access the data is to work through the low-cardinality columns with the smallest number of distinct values before the high-cardinality columns. The following sort order minimizes storage and maximizes query performance for queries that have equality restrictions on `gender`, `class`, `pass_fail`, and `name`. Specify the `ORDER BY` clause of the projection as follows:

```
ORDER BY students.gender, students.pass_fail, students.class, students.name
```

In this example, the `gender` column is represented by two RLE entries, the `pass_fail` column is represented by four entries, and the `class` column is represented by 16 entries, regardless of the cardinality of the `students` table. Vertica efficiently finds the set of rows that satisfy all the predicates, resulting in a huge reduction of search effort for RLE encoded columns that occur early in the sort order. Consequently, if you use low-cardinality columns in local predicates, as in the previous example, put those columns early in the projection sort order, in increasing order of distinct cardinality (that is, in increasing order of the number of distinct values in each column).



If you sort this table with `student.class` first, you improve the performance of queries that restrict only on the `student.class` column, and you improve the compression of the `student.class` column (which contains the largest number of distinct values), but the other columns do not compress as well. Determining which projection is better depends on the specific queries in your workload, and their relative importance.

Storage savings with compression decrease as the cardinality of the column increases; however, storage savings with compression increase as the number of bytes required to store values in that column increases.

Maximize the Advantages of RLE

To maximize the advantages of RLE encoding, use it only when the average run length of a column is greater than 10 when sorted. For example, suppose you have a table with the following columns, sorted in order of cardinality from low to high:

```
address.country, address.region, address.state, address.city, address.zipcode
```

The `zipcode` column might not have 10 sorted entries in a row with the same zip code, so there is probably no advantage to run-length encoding that column, and it could make compression worse. But there are likely to be more than 10 countries in a sorted run length, so applying RLE to the `country` column can improve performance.

Put Lower Cardinality Column First for Functional Dependencies

In general, put columns that you use for local predicates (as in the previous example) earlier in the join order to make predicate evaluation more efficient. In addition, if a lower cardinality column is uniquely determined by a higher cardinality column (like city_id uniquely determining a state_id), it is always better to put the lower cardinality, functionally determined column earlier in the sort order than the higher cardinality column.

For example, in the following sort order, the Area_Code column is sorted before the Number column in the customer_info table:

```
ORDER BY = customer_info.Area_Code, customer_info.Number, customer_info.Address
```

In the query, put the Area_Code column first, so that only the values in the Number column that start with 978 are scanned.

```
=> SELECT Address FROM customer_info WHERE Area_Code='978' AND Number='9780123457';
```

Area_code	Number	Address
508	508 012 3456 508 012 3457	1 Elm Street 1 School Street
617	617 012 3456 617 012 3457	1 Maple Avenue 1 Post Street
978	978 012 3456 978 012 3457	1 Olive Road 1 Main Street

Sort for Merge Joins

When processing a join, the Vertica optimizer chooses from two algorithms:

- **Merge join**—If both inputs are pre-sorted on the join column, the optimizer chooses a merge join, which is faster and uses less memory.
- **Hash join**—Using the hash join algorithm, Vertica uses the smaller (inner) joined table to build an in-memory hash table on the join column. A hash join has no sort requirement, but it consumes more memory because Vertica builds a hash table with the values in the inner table. The optimizer chooses a hash join when projections are not sorted on the join columns.

If both inputs are pre-sorted, merge joins do not have to do any pre-processing, making the join perform faster. Vertica uses the term sort-merge join to refer to the case when at least one of the inputs must be sorted prior to the merge join. Vertica sorts the inner input side but only if the outer input side is already sorted on the join columns.

To give the Vertica query optimizer the option to use an efficient merge join for a particular join, create projections on both sides of the join that put the join column first in their respective projections. This is primarily important to do if both tables are so large that neither table fits into memory. If all tables that a table will be joined to can be expected to fit into memory simultaneously, the benefits of merge join over hash join are sufficiently small that it probably isn't worth creating a projection for any one join column.

Sort on Columns in Important Queries

If you have an important query, one that you run on a regular basis, you can save time by putting the columns specified in the `WHERE` clause or the `GROUP BY` clause of that query early in the sort order.

If that query uses a high-cardinality column such as Social Security number, you may sacrifice storage by placing this column early in the sort order of a projection, but your most important query will be optimized.

Sort Columns of Equal Cardinality By Size

If you have two columns of equal cardinality, put the column that is larger first in the sort order. For example, a CHAR(20) column takes up 20 bytes, but an INTEGER column takes up 8 bytes. By putting the CHAR(20) column ahead of the INTEGER column, your projection compresses better.

Sort Foreign Key Columns First, From Low to High Distinct Cardinality

Suppose you have a fact table where the first four columns in the sort order make up a foreign key to another table. For best compression, choose a sort order for the fact table such that the foreign keys appear first, and in increasing order of distinct cardinality. Other factors also apply to the design of projections for fact tables, such as partitioning by a time dimension, if any.

In the following example, the table `inventory` stores inventory data, and `product_key` and `warehouse_key` are foreign keys to the `product_dimension` and `warehouse_dimension` tables:

```
=> CREATE TABLE inventory (  
  date_key INTEGER NOT NULL,  
  product_key INTEGER NOT NULL,  
  warehouse_key INTEGER NOT NULL,  
  ...  
);  
=> ALTER TABLE inventory  
  ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY(warehouse_key)  
  REFERENCES warehouse_dimension(warehouse_key);  
ALTER TABLE inventory  
  ADD CONSTRAINT fk_inventory_product FOREIGN KEY(product_key)  
  REFERENCES product_dimension(product_key);
```

The `inventory` table should be sorted by `warehouse_key` and then `product`, since the cardinality of the `warehouse_key` column is probably lower than the cardinality of the `product_key`.

Prioritizing Column Access Speed

If you measure and set the performance of storage locations within your cluster, Vertica uses this information to determine where to store columns based on their rank. For more information, see [Setting Storage Performance](#).

How Columns are Ranked

Vertica stores columns included in the projection sort order on the fastest available storage locations. Columns not included in the projection sort order are stored on slower disks.

Columns for each projection are ranked as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next-to-last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given $1000 + \#$ of sort columns.
- The remaining columns are given numbers from $1000-1$, starting with 1000 and decrementing by one per column.

Vertica then stores columns on disk from the highest ranking to the lowest ranking. It places highest-ranking columns on the fastest disks and the lowest-ranking columns on the slowest disks.

Overriding Default Column Ranking

You can modify which columns are stored on fast disks by manually overriding the default ranks for these columns. To accomplish this, set the `ACCESSRANK` keyword in the column list. Make sure to use an integer that is not already being used for another column. For example, if you want to give a column the fastest access rank, use a number that is significantly higher than `1000 + the number of sort columns`. This allows you to enter more columns over time without bumping into the access rank you set.

The following example sets column `store_key`'s access rank to 1500:

```
CREATE PROJECTION retail_sales_fact_p (  
  store_key ENCODING RLE ACCESSRANK 1500,  
  pos_transaction_number ENCODING RLE,  
  sales_dollar_amount,  
  cost_dollar_amount )  
AS SELECT  
  store_key,  
  pos_transaction_number,  
  sales_dollar_amount,  
  cost_dollar_amount  
FROM store.store_sales_fact  
ORDER BY store_key  
SEGMENTED BY HASH(pos_transaction_number) ALL NODES;
```

Managing Users and Privileges

Database users should have access to only the database resources they need to perform their tasks. For example, most users should be able to read data but not modify or insert new data, while other users might need more permissive access, such as the right to create and modify schemas, tables, and views, as well as rebalance nodes on a cluster and start or stop a database. It is also possible to allow certain users to grant other users access to the appropriate database resources.

Client authentication controls what database objects users can access and change in the database. To prevent unauthorized access, a superuser limits access to what is needed, granting privileges directly to users or to roles through a series of GRANT statements. Roles can then be granted to users, as well as to other roles.

This section introduces the [privilege role model](#) in Vertica and describes how to create and manage users.

See Also

- [About Database Privileges](#)
- [About Database Roles](#)
- [GRANT Statements](#)
- [REVOKE Statements](#)

About Database Users

Every Vertica database has one or more users. When users connect to a database, they must log on with valid credentials (username and password) that a superuser defined in the database.

Database users own the objects they create in a database, such as tables, procedures, and storage locations.

Note: By default, users have the right to [create temporary tables](#) in a database.

See Also

- [Creating a Database User](#)
- [CREATE USER](#)
- [About MC Users](#)

Types of Database Users

In an Vertica database, there are three types of users:

- Database administrator (DBADMIN)
- Object owner
- Everyone else (PUBLIC)

Note: External to a Vertica database, an MC administrator can create users through the Management Console and grant them database access. See [About MC Users](#) for details.

Database Administration User

When you install a new Vertica Analytics Platform a database administration user with access to the following roles gets created:

- [DBADMIN Role](#)
- [DBDUSER Role](#)
- [PSEUDOSUPERUSER Role](#)

Access to these roles allows this user to perform all database operations. Assign a name to this user during installation using the `--dba-user` option (use `-u` for upgrades). For example:

```
--dba-user mydba
```

This example creates a database administration user called `mydba`. The username you use here must already exist on your operating system. See [Installing Vertica with the Installation Script](#)

If you do not use `--dba-user` during installation the database administrator user gets named `DBADMIN` by default.

Note: Do not confuse the `DBADMIN` user with the [DBADMIN Role](#). The `DBADMIN` role is a set of privileges you assign to a specific user based on the user's position in your organization.

The Vertica Analytics Platform Database Administration user is also called a superuser throughout the Vertica Analytics Platform documentation. Do not confuse this superuser with the Linux superuser that manages the Linux operating system.

Create a Database Administration User in the Vertica Analytics Platform

As the Database Administration user you can create other users with the same privileges:

1. Create a user:

```
=> CREATE USER DataBaseAdmin2;  
CREATE USER
```

2. Grant the appropriate roles to the new user DataBaseAdmin2:

```
=> GRANT dbduser, dbadmin, pseudosuperuser to DataBaseAdmin2;  
GRANT ROLE
```

The user DataBaseAdmin2 now has the same privileges granted to the original Database Administration user.

3. As the DataBaseAdmin2 user, enable the roles using [SET ROLE](#):

```
=> \c VMart DataBaseAdmin2;  
You are now connected to database "VMart" as user "DataBaseAdmin2".  
=> SET ROLE dbadmin, dbduser, pseudosuperuser;  
SET ROLE
```

4. Confirm the roles are enabled:

```
=> SHOW ENABLED ROLES;  
name          | setting  
-----  
enabled roles | dbduser, dbadmin, pseudosuperuser
```

See Also

- [DBADMIN Role](#)
- [PSEUDOSUPERUSER Role](#)
- [PUBLIC Role](#)

Object Owner

An object owner is the user who creates a particular database object and can perform any operation on that object. By default, only an owner (or a superuser) can act on a database object. In order to allow other users to use an object, the owner or superuser must grant privileges to those users using one of the [GRANT Statements](#).

Note: Object owners are [PUBLIC users](#) for objects that other users own.

See [About Database Privileges](#) for more information.

PUBLIC User

All non-DBA (superuser) or object owners are PUBLIC users.

Note: Object owners are PUBLIC users for objects that other users own.

Newly-created users do not have access to schema PUBLIC by default. Make sure to GRANT USAGE ON SCHEMA PUBLIC to all users you create.

See Also

- [PUBLIC Role](#)

Creating a Database User

To create a database user:

1. From vsql, connect to the database as a superuser.
2. Issue the [CREATE USER](#) statement with optional parameters.
3. Run a series of [GRANT Statements](#) to grant the new user privileges.

To create a user on MC, see [Creating an MC User](#) in Management Console

New User Privileges

By default, new database users have the right to create temporary tables in the database.

Newly-created users do not have access to schema PUBLIC by default. Make sure to `GRANT USAGE ON SCHEMA PUBLIC` to all users you create

Modifying Users

You can change information about a user, such as his or her password, by using the [ALTER USER](#) statement. If you want to configure a user to not have any password authentication, you can set the empty password "" in `CREATE USER` or `ALTER USER` statements, or omit the `IDENTIFIED BY` parameter in `CREATE USER`.

Example

The following series of commands add user Fred to a database with password 'password'. The second command grants USAGE privileges to Fred on the public schema:

```
=> CREATE USER Fred IDENTIFIED BY 'password';  
=> GRANT USAGE ON SCHEMA PUBLIC to Fred;
```

User names created with double-quotes are case sensitive. For example:

```
=> CREATE USER "FrEd1";
```

In the above example, the logon name must be an exact match. If the user name was created without double-quotes (for example, FRED1), then the user can log on as FRED1, FrEd1, fred1, and so on.

See Also

- [Granting and Revoking Privileges](#)
- [Granting Access to Database Roles](#)

Locking/unlocking a user's Database Access

A superuser can manually lock an existing database user's account with the [ALTER USER](#) statement. For example, the following command prevents user Fred from logging in to the database:

```
=> ALTER USER Fred ACCOUNT LOCK;
```

```
=> \c - Fred
FATAL 4974: The user account "Fred" is locked
HINT: Please contact the database administrator
```

To grant Fred database access, use UNLOCK syntax with the ALTER USER command:

```
=> ALTER USER Fred ACCOUNT UNLOCK;
=> \c - Fred
You are now connected as user "Fred".
```

Using CREATE USER to lock an account

Although not as common, you can create a new user with a locked account; for example, you might want to set up an account for a user who doesn't need immediate database access, as in the case of an employee who will join the company at a future date.

```
=> CREATE USER Bob ACCOUNT UNLOCK;
CREATE USER
```

CREATE USER also supports UNLOCK syntax; however, UNLOCK is the default, so you don't need to specify the keyword when you create a new user to whom you want to grant immediate database access.

Locking an account automatically

Instead of manually locking an account, a superuser can automate account locking by setting a maximum number of failed login attempts through the CREATE PROFILE statement. See Profiles.

Changing a User's Password

A superuser can change another user's database account, including reset a password, with the [ALTER USER](#) statement.

Making changes to a database user account with does not affect current sessions.

```
=> ALTER USER Fred IDENTIFIED BY 'newpassword';
```

In the above command, Fred's password is now *newpassword*.

Note: Non-DBA users can change their own passwords using the IDENTIFIED BY 'new-password' option along with the REPLACE 'old-password' clause. See [ALTER USER](#) for details.

Changing a User's MC Password

On MC, users with ADMIN or IT privileges can reset a user's non-LDAP password from the MC interface.

Non-LDAP passwords on MC are for MC access only and are not related to a user's logon credentials on the Vertica database.

1. Sign in to Management Console and navigate to **MC Settings > User management**.
2. Click to select the user to modify and click **Edit**.
3. Click **Edit password** and enter the new password twice.
4. Click **OK** and then click **Save**.

About Database Privileges

When a database object is created, such as a schema, table, or view, that object is assigned an owner—the person who executed the CREATE statement. By default, database administrators (superusers) or object owners are the only users who can do anything with the object.

In order to allow other users to use an object, or remove a user's right to use an object, the authorized user must grant another user privileges on the object.

Privileges are granted (or revoked) through a collection of GRANT/REVOKE statements that assign the privilege—a type of permission that lets users perform an action on a database object, such as:

- Create a schema
- Create a table (in a schema)
- Create a view
- View (select) data
- Insert, update, or delete table data
- Drop tables or schemas
- Run procedures

Before Vertica executes a statement, it determines if the requesting user has the necessary privileges to perform the operation.

For more information about the privileges associated with these resources, see [Privileges That Can Be Granted on Objects](#).

Note: Vertica logs information about each grant (grantor, grantee, privilege, and so on) in the `V_CATALOG.GRANTS` system table.

See Also

- [GRANT Statements](#)
- [REVOKE Statements](#)

Inherited Privileges

Inherited privileges allow you to grant privileges at the schema level. This enables privileges to be granted automatically to new tables or views in the schema. Existing tables and views are unchanged when you alter the schema to include or exclude inherited privileges. Using inherited privileges eliminates the need to apply the same privileges to each individual table or view in the schema.

To assign inherited privileges, you must be an owner of the schema or a superuser. Assign inherited privileges using the following SQL statements

- [GRANT Statements](#)
- [CREATE SCHEMA](#)
- [ALTER SCHEMA](#)
- [CREATE TABLE](#)
- [ALTER TABLE](#)
- [CREATE VIEW](#)
- [ALTER VIEW](#)

Granting Inherited Privileges from One User to Another

The following steps describe a process for user1 to enable inherited privileges to user2.

1. The database user, user1, creates a schema (schema1), and a table (table1) in schema1:

```
user1=> CREATE SCHEMA schema1;  
user1=> CREATE TABLE schema1.table1 (id int);
```

2. User user1 grants USAGE and CREATE privileges on schema1 to user2:

```
user1=> GRANT USAGE ON SCHEMA schema1 to user2;  
user1=> GRANT CREATE ON SCHEMA schema1 to user2;
```

3. The user2 user queries schema1.table1, but the query fails:

```
user2=> SELECT * FROM schema1.table1;  
ERROR 4367: Permission denied for relation table1
```

4. The user user1 grants SELECT ON SCHEMA privilege on schema1 to user2:

```
user1=> GRANT SELECT ON SCHEMA schema1 to user2;
```

5. Next, user1 uses ALTER TABLE to include SCHEMA privileges to table1:

```
user1=> ALTER TABLE schema1.table1 INCLUDE SCHEMA PRIVILEGES;
```

6. The user2 query now succeeds:

```
user2=> SELECT * FROM schema1.table1;  
id  
---  
(0 rows)
```

7. User 1 now uses ALTER SCHEMA to include privileges so that all tables created in schema1 inherit schema privileges:

```
user1=> ALTER SCHEMA schema1 DEFAULT INCLUDE PRIVILEGES;  
user1=> CREATE TABLE schema1.table2 (id int);
```

8. With Inherited Privileges enabled, user2 can query table2 without user1 having to specifically grant privileges on table2:

```
user2=> SELECT * FROM schema1.table2;  
id  
---  
(0 rows)
```

Enable or Disable Inherited Privileges at the Database Level

Use the `disableinheritedprivileges` configuration parameter to enable (0) Inherited Privileges:

```
=> ALTER DATABASE [database name] SET disableinheritedprivileges = 0;
```

Use the following configuration parameter to disable (1) Inherited Privileges:

```
=> ALTER DATABASE [database name] SET disableinheritedprivileges = 1;
```

Grant Inherited Privileges

Grant inherited privileges at the schema level. When inherited privileges is enabled, all privileges granted to the schema are automatically granted to all newly created tables or views in the schema. Existing tables or views remain unchanged when you alter the schema to include or exclude inherited privileges.

By default, inherited privileges are enabled at the database level and disabled at the schema level (unless you indicate otherwise while running `CREATE SCHEMA`). See [Enable or Disable Inherited Privileges at the Database Level](#) for more information. To apply inherited privileges, you must meet one of the following conditions:

- Be the owner of the object
- Be a superuser

Inherit Privileges on a Schema

Use the [CREATE SCHEMA](#) or [ALTER SCHEMA](#) SQL statements to apply inherited privileges to a schema. The tables and views in that schema then inherit any privileges granted to the schema by default.

This example shows how to create a new schema with inherited privileges. The `DEFAULT` parameter sets the default behavior so that all new tables and views created in this schema automatically inherit the schema's privileges:

```
=> CREATE SCHEMA s1 DEFAULT INCLUDE PRIVILEGES;
```

This example shows how to modify an existing schema to enable inherited privileges:

```
=> ALTER SCHEMA s1 DEFAULT INCLUDE PRIVILEGES;
```

Note: After enabling inherited privileges on a schema, you still must grant privileges on the schema to a user or role. From the [GRANT \(Schema\)](#) statement, use the following parameters with inherited privileges enabled:

- SELECT
- INSERT
- UPDATE

- DELETE
- REFERENCES
- TRUNCATE

The following message appears when you specify `INCLUDE PRIVILEGES` while Inherited privileges is disabled at the database level:

```
Inherited privileges are globally disabled; schema parameter is set but has no effect.
```

See [Enable or Disable Inherited Privileges at the Database Level](#) to enable Inherited Privileges at the database level.

Inherit Privileges on a Table or Flex Table

You can specify an individual table or flex table to inherit privileges from the schema. Use [CREATE TABLE](#) or [ALTER TABLE](#) SQL statements to enable inherited privileges for a table. The table-level flag takes priority over the schema flag, while the database knob takes priority over both.

This example shows creating a new table with inherited privileges:

```
=> CREATE TABLE s1.t1 ( x int) INCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing table to enable inherited privileges:

```
=> ALTER TABLE s1.t1 INCLUDE SCHEMA PRIVILEGES;
```

If you run [CREATE TABLE](#), [CREATE TABLE LIKE](#), or [CREATE TABLE AS SELECT](#) in a schema with inherited privileges set, the following informational warning appears:

```
=> CREATE TABLE s1.t1 ( x int);  
WARNING: Table <table_name> will include privileges from schema <schema_name>
```

Note that this message does not appear when you add the `INCLUDE SCHEMA PRIVILEGES` statement.

Exclude Privileges on a Table

You can exclude a table in a schema with inherited privileges so that table does not inherit the schema's privileges. Use [CREATE TABLE](#) or [ALTER TABLE](#) SQL statements to exclude inherited

privileges for a table.

This example shows creating a new table and excluding schema privileges:

```
=> CREATE TABLE s1.t1 ( x int) EXCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing table to exclude inherited privileges:

```
=> ALTER TABLE s1.t1 EXCLUDE SCHEMA PRIVILEGES;
```

Include Privileges on a View

You can specify a View to inherit privileges from the schema. Use [CREATE VIEW](#) or [ALTER VIEW](#) SQL statements to enable inherited privileges for a view.

This example shows creating a view with inherited privileges enabled:

```
=> CREATE VIEW view1 INCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing view to enable inherited privileges:

```
=> ALTER VIEW veiw1 INCLUDE SCHEMA PRIVILEGES;
```

Exclude Privileges on a View

You can exclude a view in a schema with inherited privileges so that view does not inherit the schema's privileges. Use [CREATE VIEW](#) or [ALTER VIEW](#) SQL statements to exclude inherited privileges for a view.

This example shows creating a new view and excluding schema privileges:

```
=> CREATE VIEW view1 EXCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing view to exclude inherited privileges:

```
=> ALTER VIEW view1 EXCLUDE SCHEMA PRIVILEGES;
```

Default Privileges for All Users

To set the minimum level of privilege for all users, Vertica has the special [PUBLIC Role](#), which it grants to each user automatically. This role is automatically enabled, but the database

administrator or a superuser can also grant higher privileges to users separately using GRANT statements.

The following topics discuss those higher privileges.

Default Privileges for MC Users

Privileges on Management Console (MC) are managed through roles, which determine a user's access to MC and to MC-managed Vertica databases through the MC interface. MC privileges do not alter or override Vertica privileges or roles. See [About MC Privileges and Roles](#) for details.

Privileges Required for Common Database Operations

This topic lists the required privileges for database objects in Vertica.

Unless otherwise noted, superusers can perform all of the operations shown in the following tables without any additional privilege requirements. Object owners have the necessary rights to perform operations on their own objects, by default.

Schemas

The PUBLIC schema is present in any newly-created Vertica database, and newly-created users have only USAGE privilege on PUBLIC. A database superuser must explicitly grant new users CREATE privileges, as well as grant them individual object privileges so the new users can create or look up objects in the PUBLIC schema.

Operation	Required Privileges
CREATE SCHEMA	CREATE privilege on database
DROP SCHEMA	Schema owner
ALTER SCHEMA RENAME	CREATE privilege on database

Tables

Operation	Required Privileges
CREATE TABLE	CREATE privilege on schema

Operation	Required Privileges
	<p>Note: Referencing sequences in the CREATE TABLE statement requires the following privileges:</p> <ul style="list-style-type: none"> • SELECT privilege on sequence object • USAGE privilege on sequence schema
DROP TABLE	USAGE privilege on the schema that contains the table or schema owner
TRUNCATE TABLE	USAGE privilege on the schema that contains the table or schema owner
ALTER TABLE ADD/DROP/ RENAME/ALTER-TYPE COLUMN	USAGE privilege on the schema that contains the table
ALTER TABLE ADD/DROP CONSTRAINT	USAGE privilege on the schema that contains the table
ALTER TABLE PARTITION (REORGANIZE)	USAGE privilege on the schema that contains the table
ALTER TABLE RENAME	USAGE and CREATE privilege on the schema that contains the table
ALTER TABLE SET SCHEMA	<ul style="list-style-type: none"> • CREATE privilege on new schema • USAGE privilege on the old schema
SELECT	<ul style="list-style-type: none"> • SELECT privilege on table • USAGE privilege on schema that contains the table
INSERT	<ul style="list-style-type: none"> • INSERT privilege on table • USAGE privilege on schema that contains the table
DELETE	<ul style="list-style-type: none"> • DELETE privilege on table • USAGE privilege on schema that contains the table • SELECT privilege on the referenced table when executing a DELETE statement that references table column values in a WHERE or SET clause
UPDATE	<ul style="list-style-type: none"> • UPDATE privilege on table

Operation	Required Privileges
	<ul style="list-style-type: none"> • USAGE privilege on schema that contains the table • SELECT privilege on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause
REFERENCES	<ul style="list-style-type: none"> • REFERENCES privilege on table to create foreign key constraints that reference this table • USAGE privileges on schema that contains the constrained table and the source of the foreign k
ANALYZE_STATISTICS	<ul style="list-style-type: none"> • INSERT/UPDATE/DELETE privilege on table • USAGE privilege on schema that contains the table
DROP_STATISTICS	<ul style="list-style-type: none"> • INSERT/UPDATE/DELETE privilege on table • USAGE privilege on schema that contains the table
DROP_PARTITION	USAGE privilege on schema that contains the table

Views

Operation	Required Privileges
CREATE VIEW	<ul style="list-style-type: none"> • CREATE privilege on the schema to contain a view • SELECT privileges on base objects (tables/views) • USAGE privileges on schema that contains the base objects
DROP VIEW	USAGE privilege on schema that contains the view or schema owner
SELECT ... FROM VIEW	<ul style="list-style-type: none"> • SELECT privilege on view • USAGE privilege on the schema that contains the view. • View owner must have SELECT ... WITH GRANT OPTION privileges on the view's anchor tables or views if non-owner runs a SELECT query on the view.

Operation	Required Privileges
	<ul style="list-style-type: none"> View owner must have SELECT privilege on a view's base objects (table or view) if owner runs a SELECT query on the view.

Projections

Operation	Required Privileges
CREATE PROJECTION	<ul style="list-style-type: none"> SELECT privilege on anchor tables USAGE privilege on schema that contains anchor tables or schema owner CREATE privilege on schema to contain the projection <p>Note: If a projection is implicitly created with the table, no additional privilege is needed other than privileges for table creation.</p>
AUTO/DELAYED PROJECTION	<p>On projections created during INSERT..SELECT or COPY operations:</p> <ul style="list-style-type: none"> SELECT privilege on anchor tables USAGE privilege on schema that contains anchor tables
ALTER PROJECTION RENAME	USAGE and CREATE privilege on schema that contains the projection
DROP PROJECTION	USAGE privilege on schema that contains the projection or schema owner

External Procedures

Operation	Required Privileges
CREATE PROCEDURE	Superuser
DROP PROCEDURE	Superuser
EXECUTE	<ul style="list-style-type: none"> EXECUTE privilege on procedure

Operation	Required Privileges
	<ul style="list-style-type: none"> • USAGE privilege on schema that contains the procedure

Libraries

Operation	Required Privileges
<code>CREATE LIBRARY</code>	Superuser
<code>DROP LIBRARY</code>	Superuser

User-Defined Functions

The following abbreviations are used in the UDF table:

- UDF = Scalar
- UDT = Transform
- UDAnF= Analytic
- UDAF = Aggregate

Operation	Required Privileges
<code>CREATE FUNCTION (SQL)</code> <code>CREATE FUNCTION (UDF)</code> <code>CREATE TRANSFORM FUNCTION (UDF)</code> <code>CREATE ANALYTIC FUNCTION (UDAnF)</code> <code>CREATE AGGREGATE FUNCTION (UDAF)</code>	<ul style="list-style-type: none"> • CREATE privilege on schema to contain the function • USAGE privilege on base library (if applicable)
<code>DROP FUNCTION</code> <code>DROP TRANSFORM FUNCTION</code> <code>DROP ANALYTIC FUNCTION</code> <code>DROP AGGREGATE FUNCTION</code>	<ul style="list-style-type: none"> • Superuser or function owner • USAGE privilege on schema that contains the function
<code>ALTER FUNCTION (UDF or UDT) RENAME TO</code>	USAGE and CREATE privilege on schema that contains the function
<code>ALTER FUNCTION (UDF or UDT) SET SCHEMA</code>	<ul style="list-style-type: none"> • USAGE privilege on schema that currently

Operation	Required Privileges
	<p>contains the function (old schema)</p> <ul style="list-style-type: none"> • CREATE privilege on the schema to which the function will be moved (new schema)
EXECUTE (SQL/UDF/UDT/ ADAF/UDAnF) function	<ul style="list-style-type: none"> • EXECUTE privilege on function • USAGE privilege on schema that contains the function

Sequences

Operation	Required Privileges
CREATE SEQUENCE	<p>CREATE privilege on schema to contain the sequence</p> <p>Note: Referencing sequence in the CREATE TABLE statement requires SELECT privilege on sequence object and USAGE privilege on sequence schema.</p>
CREATE TABLE with SEQUENCE	<ul style="list-style-type: none"> • SELECT privilege on sequence • USAGE privilege on sequence schema
DROP SEQUENCE	<p>USAGE privilege on schema containing the sequence or schema owner</p>
ALTER SEQUENCE RENAME TO	<p>USAGE and CREATE privileges on schema</p>
ALTER SEQUENCE SET SCHEMA	<ul style="list-style-type: none"> • USAGE privilege on the schema that currently contains the sequence (old schema) • CREATE privilege on new schema to contain the sequence
CURRVAL()NEXTVAL()	<ul style="list-style-type: none"> • SELECT privilege on sequence • USAGE privilege on sequence schema

Resource Pools

Operation	Required Privileges
CREATE RESOURCE POOL	Superuser
ALTER RESOURCE POOL	<p>Superuser on the resource pool to alter:</p> <ul style="list-style-type: none"> • MAXMEMORYSIZE • PRIORITY • QUEUETIMEOUT <p>UPDATE privilege on the resource pool to alter:</p> <ul style="list-style-type: none"> • PLANNEDCONCURRENCY • SINGLEINITIATOR • MAXCONCURRENCY
SET SESSION RESOURCE_POOL	<ul style="list-style-type: none"> • USAGE privilege on the resource pool • Users can only change their own resource pool setting using ALTER USER syntax
DROP RESOURCE POOL	Superuser

Users/Profiles/Roles

Operation	Required Privileges
CREATE USER CREATE PROFILE CREATE ROLE	Superuser
ALTER USER ALTER PROFILE ALTER ROLE RENAME	Superuser
DROP USER DROP PROFILE	Superuser

Operation	Required Privileges
DROP ROLE	

Object Visibility

You can use one or a combination of vsql [\d \[pattern\]](#) meta commands and [SQL system tables](#) to view objects on which you have privileges to view.

- Use [\dn \[pattern\]](#) to view schema names and owners
- Use [\dt \[pattern\]](#) to view all tables in the database, as well as the system table [V_CATALOG.TABLES](#)
- Use [\dj \[pattern\]](#) to view projections showing the schema, projection name, owner, and node, as well as the system table [V_CATALOG.PROJECTIONS](#)

Operation	Required Privileges
Look up schema	At least one privilege on schema that contains the object
Look up Object in Schema or in System Tables	USAGE privilege on schema At least one privilege on any of the following objects: TABLE VIEW FUNCTION PROCEDURE SEQUENCE

Operation	Required Privileges
Look up Projection	At least one privilege on all anchor tables USAGE privilege on schema of all anchor table
Look up resource pool	SELECT privilege on the resource pool
Existence of object	USAGE privilege on the schema that contains the object

I/O Operations

Operation	Required Privileges
CONNECT/DISCONNECT	None
EXPORT TO Vertica	<ul style="list-style-type: none"> • SELECT privileges on the source table • USAGE privilege on source table schema • INSERT privileges for the destination table in target database • USAGE privilege on destination table schema
COPY FROM Vertica	<ul style="list-style-type: none"> • SELECT privileges on the source table • USAGE privilege on source table schema • INSERT privileges for the destination table in target

Operation	Required Privileges
	database <ul style="list-style-type: none"> • USAGE privilege on destination table schema
<code>COPY FROM file</code>	Superuser
<code>COPY FROM STDIN</code>	<ul style="list-style-type: none"> • INSERT privilege on table • USAGE privilege on schema
<code>COPY LOCAL</code>	<ul style="list-style-type: none"> • INSERT privilege on table • USAGE privilege on schema

Comments

Operation	Required Privileges
COMMENT ON { is one of }: <ul style="list-style-type: none"> • AGGREGATE FUNCTION • ANALYTIC FUNCTION • COLUMN • CONSTRAINT • FUNCTION • LIBRARY • NODE • PROJECTION • SCHEMA • SEQUENCE • TABLE • TRANSFORM FUNCTION 	Object owner or superuser

Operation	Required Privileges
<ul style="list-style-type: none">VIEW	

Transactions

Operation	Required Privileges
COMMIT	None
ROLLBACK	None
RELEASE SAVEPOINT	None
SAVEPOINT	None

Sessions

Operation	Required Privileges
SET { is one of }: <ul style="list-style-type: none">DATESTYLEESCAPE_STRING_WARNINGINTERVALSTYLELOCALEROLESEARCH_PATHSESSION AUTOCOMMITSESSION CHARACTERISTICSSESSION MEMORYCAPSESSION RESOURCE POOLSESSION RUNTIMECAP	None

Operation	Required Privileges
<ul style="list-style-type: none">• SESSION TEMPSPACE• STANDARD_CONFORMING_STRINGS• TIMEZONE	
SHOW { name ALL }	None

Tuning Operations

Operation	Required Privileges
PROFILE	Same privileges required to run the query being profiled
EXPLAIN	Same privileges required to run the query for which you use the EXPLAIN keyword

Privileges That Can Be Granted on Objects

The following table provides an overview of privileges that can be granted on (or revoked from) database objects in Vertica:

	Cre ate	Usa ge	Execu te	Sele ct	Inse rt	Dele te	Upda te	Referen ces	Rea d	Writ e	Trunc ate
Database	Y										
Schem a	Y	Y		Y	Y	Y	Y	Y			Y
Table				Y	Y	Y	Y	Y			Y
View				Y							
Sequen ce				Y							
Proced ure			Y								
UDx			Y								
Library		Y									
Resour ce Pool		Y									
Storage Locatio n									Y	Y	

See Also

- [GRANT Statements](#)
- [REVOKE Statements](#)

Database Privileges

Only a database superuser can create a database. In a new database, the [PUBLIC Role](#) is granted USAGE on the automatically-created PUBLIC schema. It is up to the superuser to grant further privileges to users and roles.

The only privilege a superuser can grant on the database itself is CREATE, which allows the user to create a new schema in the database. For details on granting and revoking privileges on a database, see the [GRANT \(Database\)](#) and [REVOKE \(Database\)](#) topics in the SQL Reference Manual.

Privilege	Grantor	Description
CREATE	Superuser	Allows a user to create a schema.

Schema Privileges

By default, only a superuser and the schema owner have privileges to create objects within a schema. Additionally, only the schema owner or a superuser can drop or alter a schema. See [DROP SCHEMA](#) and [ALTER SCHEMA](#).

You must grant all new users access to the PUBLIC schema by running GRANT USAGE ON SCHEMA PUBLIC. Then grant new users CREATE privileges and privileges to individual objects in the schema. This enables new users to create or locate objects in the PUBLIC schema. Without USAGE privilege, objects in the schema cannot be used or altered, even by the object owner.

CREATE gives the schema owner or user WITH GRANT OPTION permission to create new objects in the schema, including renaming an object in the schema or moving an object into this schema.

Note: The schema owner is typically the user who creates the schema. However, a superuser can create a schema and assign ownership of the schema to a different user at creation.

All other access to the schema and its objects must be explicitly granted to users or roles by the superuser or schema owner. This prevents unauthorized users from accessing the schema and its objects. A user can be granted one of the following privileges through the GRANT statement:

Privilege	Description
CREATE	Allows the user to create new objects within the schema. This includes the ability to create a new object, rename existing objects, and move objects into the schema from other schemas.
USAGE	Permission to select, access, alter, and drop objects in the schema. The user must also be granted access to the individual objects in order to alter them. For example, a user would need to be granted USAGE on the schema and SELECT on a table to be able to select data from a table. You receive an error message if you attempt to query a table that you have SELECT privileges on, but do not have USAGE privileges for the schema that contains the table.

Note the following on error messages related to granting privileges on a schema or an object:

- You attempt to grant a privilege to a schema, but you do not have USAGE privilege for the schema. In this case, you receive an error message that the schema does not exist.
- You attempt to grant a privilege to an object within a schema, and you have USAGE privilege on the schema. You do not have privilege on the individual object within the schema. In this case, you receive an error denying permission for that object.

Schema Privileges and the Search Path

The search path determines to which schema unqualified objects in SQL statements belong.

When a user specifies an object name in a statement without supplying the schema in which the object exists (called an unqualified object name) Vertica has two different behaviors, depending on whether the object is being accessed or created.

Creating an object	Accessing/altering an object
<p>When a user creates an object—such as table, view, sequence, procedure, function—with an unqualified name, Vertica tries to create the object in the current schema (the first schema in the schema search path), returning an error if the schema does not exist or if the user does not have CREATE privileges in that schema.</p> <p>Use the SHOW search_path command to view the current search path.</p> <pre>=> SHOW search_path; name setting -----+----- search_path "\$user", public, v_catalog, v_monitor, v_internal (1 row)</pre>	<p>When a user accesses or alters an object with an unqualified name, those statements search through all schemas for a matching object, starting with the current schema, where:</p> <ul style="list-style-type: none"> • The object name in the schema matches the object name in the

Creating an object	Accessing/altering an object
<p>Note: The first schema in the search path is the current schema, and the \$user setting is a placeholder that resolves to the current user's name.</p>	<p>statement.</p> <ul style="list-style-type: none"> • The user has USAGE privileges on the schema in order to access object in it. • The user has at least one privilege on the object.

See Also

- [Setting Search Paths](#)
- [GRANT \(Schema\)](#)
- [REVOKE \(Schema\)](#)

Table Privileges

By default, only a superuser and the table owner (typically the person who creates a table) have access to a table. The ability to drop or alter a table is also reserved for a superuser or table owner. This privilege cannot be granted to other users.

All other users or roles (including the user who owns the schema, if he or she does not also own the table) must be explicitly granted using WITH GRANT OPTION syntax to access the table.

These are the table privileges a superuser or table owner can grant:

Privilege	Description
SELECT	Permission to run SELECT queries on the table.
INSERT	Permission to INSERT data into the table.
DELETE	Permission to DELETE data from the table, as well as SELECT privilege on the table when executing a DELETE statement that references table column values in a WHERE or SET clause.
UPDATE	Permission to UPDATE and change data in the table, as well as SELECT privilege

Privilege	Description
	on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause.
REFERENCES	Permission to CREATE foreign key constraints that reference this table.

To use any of the above privileges, the user must also have USAGE privileges on the schema that contains the table. See [Schema Privileges](#) for details.

Referencing sequence in the [CREATE TABLE](#) statement requires the following privileges:

- SELECT privilege on sequence object
- USAGE privilege on sequence schema

For details on granting and revoking table privileges, see [GRANT \(Table\)](#) and [REVOKE \(Table\)](#) in the SQL Reference Manual.

Projection Privileges

Because projections are the underlying storage construct for tables, they are atypical in that they do not have an owner or privileges associated with them directly. Instead, the privileges to create, access, or alter a projection are based on the anchor tables that the projection references, as well as the schemas that contain them.

All queries in Vertica obtain data from projections directly or indirectly. In both cases, to run a query, you must have SELECT privileges on the tables that the projections reference, and USAGE privileges on all schemas that contain those tables.

You can create projections in two ways: explicitly and implicitly.

Explicit Projection Creation

To create a projection with [CREATE PROJECTION](#) or any of its variants, you must be a superuser or owner of the anchor table.

Only the anchor table owner can drop explicitly created projections. Explicitly created projections can be live aggregate projections, including Top-K projections and projections with expressions.

Implicit Projection Creation

When you insert data into a table, Vertica automatically creates a superprojection for the table.

Superprojections do not require any additional privileges to create or drop, other than privileges for table creation. Users who can create a table or drop a table can also create and drop the associated superprojection.

View Privileges

A view is a stored query that dynamically accesses and computes data from the database at execution time. Use `\dV` in `vsq` to display available views. By default, only the following users have privileges to access a view's base object:

- Superuser
- View owner—typically, the view creator

To execute a query that contains a view, you must have:

- SELECT privileges assigned with [GRANT \(View\)](#)
- USAGE privileges on the view's schema, assigned with [GRANT \(Schema\)](#).

You can assign view privileges to other users and roles using [GRANT \(View\)](#). For example:

- Assign `GRANT ALL` privileges to a user or role.

```
=> GRANT all privileges on view1 to role1 with grant option;
```

- Assign `GRANT ROLE` privileges to a specific role to provide view privileges. In the following example, privileges that are assigned to `role1` are assigned to `role2`:

```
=> CREATE ROLE role1;  
=> CREATE ROLE role2;  
=> GRANT role1 to role2;
```

See Also

[GRANT \(View\)](#)

[REVOKE \(View\)](#)

Sequence Privileges

To create a sequence, a user must have `CREATE` privileges on a schema that contains the sequence. Only the owner and superusers can initially access the sequence. All other users must be granted access to the sequence by a superuser or the owner.

Only the sequence owner (typically the person who creates the sequence) or the superuser can drop or rename a sequence, or change the schema in which the sequence resides:

- [DROP SEQUENCE](#): Only a sequence owner or schema owner can drop a sequence.
- [ALTER SEQUENCE RENAME TO](#): A sequence owner must have `USAGE` and `CREATE` privileges on the schema that contains the sequence to be renamed.
- [ALTER SEQUENCE SET SCHEMA](#): A sequence owner must have `USAGE` privilege on the schema that currently contains the sequence (old schema), as well as `CREATE` privilege on the schema where the sequence will be moved (new schema).

The following table lists the privileges that can be granted to users or roles on sequences.

The only privilege that can be granted to a user or role is `SELECT`, which allows the user to use [CURRVAL\(\)](#) and [NEXTVAL\(\)](#) on sequence and reference in table. The user or role also needs to have `USAGE` privilege on the schema containing the sequence.

Privilege	Description
<code>SELECT</code>	Permission to use CURRVAL() and NEXTVAL() on sequence and reference in table.
<code>USAGE</code>	Permissions on the schema that contains the sequence.

Note: Referencing sequence in the [CREATE TABLE](#) statement requires `SELECT` privilege on sequence object and `USAGE` privilege on sequence schema.

For details on granting and revoking sequence privileges, see [GRANT \(Sequence\)](#) and [REVOKE \(Sequence\)](#) in the SQL Reference Manual.

See Also

- [Working with Sequence Types](#)

External Procedure Privileges

Only a superuser is allowed to create or drop an external procedure.

By default, users cannot execute external procedures. A superuser must grant users and roles this right, using the `GRANT (Procedure) EXECUTE` statement. Additionally, users must have `USAGE` privileges on the schema that contains the procedure in order to call it.

Privilege	Description
EXECUTE	Permission to run an external procedure.
USAGE	Permission on the schema that contains the procedure.

For details on granting and revoking external table privileges, see [GRANT \(Procedure\)](#) and [REVOKE \(Procedure\)](#) in the SQL Reference Manual.

User-Defined Function Privileges

User-defined functions (described in [CREATE FUNCTION Statements](#)) can be created by superusers or users with `CREATE` privileges on the schema that will contain the function, as well as `USAGE` privileges on the base library (if applicable).

Users or roles other than the function owner can use a function only if they have been granted `EXECUTE` privileges on it. They must also have `USAGE` privileges on the schema that contains the function to be able to call it.

Privilege	Description
EXECUTE	Permission to call a user-defined function.
USAGE	Permission on the schema that contains the function.

- [DROP FUNCTION](#): Only a superuser or function owner can drop the function.
- [ALTER FUNCTION \(UDF or UDT\) RENAME TO](#): A superuser or function owner must have `USAGE` and `CREATE` privileges on the schema that contains the function to be renamed.
- [ALTER FUNCTION \(UDF or UDT\) SET SCHEMA](#): A superuser or function owner must have `USAGE` privilege on the schema that currently contains the function (old schema), as well as `CREATE` privilege on the schema where the function will be moved (new schema).

For details on granting and revoking user-defined function privileges, see the following topics in the SQL Reference Manual:

- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)

Library Privileges

Only a superuser can load an external library using the [CREATE LIBRARY](#) statement. By default, only a superuser can create user-defined functions (UDFs) based on a loaded library. A superuser can use the `GRANT USAGE ON LIBRARY` statement to allow users to create UDFs based on classes in the library. The user must also have `CREATE` privileges on the schema that will contain the UDF.

Privilege	Description
USAGE	Permission to create UDFs based on classes in the library

Once created, only a superuser or the user who created a UDF can use it by default. Either of them can grant other users or roles the ability to call the function using the `GRANT EXECUTE ON FUNCTION` statement. See the [GRANT \(User Defined Extension\)](#) and [REVOKE \(User Defined Extension\)](#) topics in the SQL Reference Manual for more information on granting and revoking privileges on functions.

In addition to `EXECUTE` privilege, users/roles also require `USAGE` privilege on the schema in which the function resides in order to execute the function.

For more information about libraries and UDFs, see [Developing User-Defined Extensions \(UDxs\)](#) in *Extending Vertica*.

Resource Pool Privileges

Only a superuser can create, alter, or drop a resource pool.

By default, users are granted `USAGE` rights to the `GENERAL` pool, from which their queries and other statements allocate memory and get their priorities. A superuser must grant users `USAGE` rights to any additional resource pools by using the `GRANT USAGE ON RESOURCE POOL` statement. Once granted access to the resource pool, users can use the [SET SESSION RESOURCE_POOL](#) statement and the `RESOURCE POOL` clause of the [ALTER USER](#) statement to have their queries draw their resources from the new pool.

Privilege	Description
USAGE	Permission to use a resource pool.
SELECT	Permission to look up resource pool information/status in system tables.
UPDATE	Permission to adjust the tuning parameters of the pool.

For details on granting and revoking resource pool privileges, see [GRANT \(Resource Pool\)](#) and [REVOKE \(Resource Pool\)](#) in the SQL Reference Manual.

Storage Location Privileges

Users and roles without superuser privileges can copy data to and from storage locations as long as the following conditions are met, where a superuser:

1. Creates a special class of storage location ([CREATE LOCATION](#)) specifying the USAGE argument set to 'USER' , which indicates the specified area is accessible to non-superusers users.
2. Grants users or roles READ and/or WRITE access to the specified location using the [GRANT \(Storage Location\)](#) statement.

Note: GRANT/REVOKE (Storage Location) statements are applicable only to 'USER' storage locations.

Once such storage locations exist and the appropriate privileges are granted, users and roles granted READ privileges can copy data from files in the storage location into a table. Those granted WRITE privileges can export data from a table to the storage location on which they have been granted access. WRITE privileges also let users save COPY statement exceptions and rejected data files from Vertica to the specified storage location.

Only a superuser can add, alter, retire, drop, and restore a location, as well as set and measure location performance. All non-dbadmin users or roles require READ and/or WRITE permissions on the location.

Privilege	Description
READ	Allows the user to copy data from files in the storage location into a table.
WRITE	Allows the user to copy data to the specific storage location. Users with WRITE privileges can also save COPY statement exceptions and rejected data files to the specified storage location.

See Also

- [GRANT \(Storage Location\)](#)
- [Storage Management Functions](#)
- [CREATE LOCATION](#)

Role, profile, and User Privileges

Only a superuser can create, alter or drop a:

- role
- profile
- user

By default, only the superuser can grant or revoke a role to another user or role. A user or role can be given the privilege to grant and revoke a role by using the WITH ADMIN OPTION clause of the GRANT statement.

For details on granting and revoking role privileges, see [GRANT \(Role\)](#) and [REVOKE \(Role\)](#) in the SQL Reference Manual.

See Also

- [CREATE USER](#)
- [ALTER USER](#)
- [DROP USER](#)
- [CREATE PROFILE](#)
- [ALTER PROFILE](#)
- [DROP PROFILE](#)
- [CREATE ROLE](#)

- [ALTER ROLE RENAME](#)
- [DROP ROLE](#)

Metadata Privileges

A superuser has unrestricted access to all database metadata. Other users have significantly reduced access to metadata based on their privileges, as follows:

Type of Metadata	User Access
<p>Catalog objects:</p> <ul style="list-style-type: none">• Tables• Columns• Constraints• Sequences• External Procedures• Projections• ROS containers• WOS	<p>Users must possess USAGE privilege on the schema and any type of access (SELECT) or modify privilege on the object to see catalog metadata about the object. See also Schema Privileges.</p> <p>For internal objects like projections, WOS and ROS containers that don't have access privileges directly associated with them, the user must possess the requisite privileges on the associated schema and table objects instead. For example, to see whether a table has any data in the WOS, you need to have USAGE on the table schema and at least SELECT on the table itself. See also Table Privileges and Projection Privileges.</p>
<p>User sessions and functions, and system tables related to these sessions</p>	<p>Users can only access information about their own, current sessions.</p> <p>The following functions provide restricted functionality to users:</p> <ul style="list-style-type: none">• CURRENT_DATABASE• CURRENT_SCHEMA• CURRENT_USER• HAS_TABLE_PRIVILEGE

Type of Metadata	User Access
	<ul style="list-style-type: none">• SESSION_USER (same as CURRENT_USER) <p>The system table, SESSIONS, provides restricted functionality to users.</p>
Storage locations	<p>Users require READ permissions to copy data from storage locations.</p> <p>Only a superuser can add or retire storage locations.</p>

I/O Privileges

Users need no special permissions to connect to and disconnect from an Vertica database.

To [EXPORT TO](#) and [COPY FROM](#) Vertica, the user must have:

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema

To [COPY FROM STDIN](#) and use local [COPY](#) a user must have INSERT privileges on the table and USAGE privilege on schema.

Note: Only a superuser can [COPY](#) from *file*.

Comment Privileges

A comment lets you add, revise, or remove a textual message to a database object. You must be an object owner or superuser in order to [COMMENT ON](#) one of the following objects:

- [COLUMN](#)
- [CONSTRAINT](#)
- [FUNCTION](#) (including AGGREGATE and ANALYTIC)

- [LIBRARY](#)
- [NODE](#)
- [PROJECTION](#)
- [SCHEMA](#)
- [SEQUENCE](#)
- [TABLE](#)
- [TRANSFORM FUNCTION](#)
- [VIEW](#)

Other users must have VIEW privileges on an object to view its comments.

Transaction Privileges

No special permissions are required for the following database operations:

- [COMMIT](#)
- [ROLLBACK](#)
- [RELEASE SAVEPOINT](#)
- [SAVEPOINT](#)

Session Privileges

No special permissions are required for users to use the SHOW statement or any of the SET statements.

Tuning Privileges

In order to [PROFILE](#) a single SQL statement or returns a query plan's execution strategy to standard output using the [EXPLAIN](#) command, users must have the same privileges that are required for them to run the same query without the PROFILE or EXPLAIN keyword.

Granting and Revoking Privileges

To grant or revoke a privilege using one of the SQL GRANT or REVOKE statements, the user must have the following permissions for the GRANT/REVOKE statement to succeed:

- Superuser or privilege WITH GRANT OPTION
- USAGE privilege on the schema
- Appropriate privileges on the object

The syntax for granting and revoking privileges is different for each database object, such as schema, database, table, view, sequence, procedure, function, resource pool, and so on.

Normally, a superuser first [creates a user](#) and then uses GRANT syntax to define the user's privileges or roles or both. For example, the following series of statements creates user Carol and grants Carol access to the apps database in the PUBLIC schema and also lets Carol grant SELECT privileges to other users on the applog table:

```
=> CREATE USER Carol;  
=> GRANT USAGE ON SCHEMA PUBLIC to Carol;  
=> GRANT ALL ON DATABASE apps TO Carol;  
=> GRANT SELECT ON applog TO Carol WITH GRANT OPTION;
```

See [GRANT Statements](#) and [REVOKE Statements](#) in the SQL Reference Manual.

About Superuser Privileges

A superuser (DBADMIN) is the automatically-created database user who has the same name as the Linux database administrator account and who can bypass all GRANT/REVOKE authorization, as well as supersede any user that has been granted the [PSEUDOSUPERUSER](#) role.

Note: Database superusers are not the same as a Linux superuser with (root) privilege and cannot have Linux superuser privilege.

A superuser can grant privileges on all database object types to other users, as well as grant privileges to roles. Users who have been granted the role will then gain the privilege as soon as they [enable it](#).

Superusers may grant or revoke any object privilege on behalf of the object owner, which means a superuser can grant or revoke the object privilege if the object owner could have

granted or revoked the same object privilege. A superuser may revoke the privilege that an object owner granted, as well as the reverse.

Since a superuser is acting on behalf of the object owner, the GRANTOR column of [V_CATALOG.GRANTS](#) table displays the object owner rather than the superuser who issued the GRANT statement.

A superuser can also alter ownership of table and sequence objects.

See Also

[DBADMIN Role](#)

About Schema Owner Privileges

By default, the schema owner has privileges to create objects within a schema. Additionally, the schema owner can drop any object in the schema, requiring no additional privilege on the object.

The schema owner is typically the user who creates the schema.

Schema owners cannot access objects in the schema. Access to objects requires the appropriate privilege at the object level.

All other access to the schema and its objects must be explicitly granted to users or roles by a superuser or schema owner to prevent unauthorized users from accessing the schema and its objects.

See [Schema Privileges](#)

About Object Owner Privileges

The database, along with every object in it, has an owner. The object owner is usually the person who created the object, although a superuser can alter ownership of objects, such as table and sequence.

Object owners must have appropriate schema privilege to access, alter, rename, move or drop any object it owns without any additional privileges.

An object owner can also:

- **Grant privileges on their own object to other users**

The WITH GRANT OPTION clause specifies that a user can grant the permission to other users. For example, if user Bob creates a table, Bob can grant privileges on that table to users Ted, Alice, and so on.

- **Grant privileges to roles**

Users who are granted the role gain the privilege.

How to Grant Privileges

As described in [Granting and Revoking Privileges](#), specific users grant privileges using the GRANT statement with or without the optional WITH GRANT OPTION, which allows the user to grant the same privileges to other users.

- A superuser can grant privileges on all object types to other users.
- A superuser or object owner can grant privileges to roles. Users who have been granted the role then gain the privilege.
- An object owner can grant privileges on the object to other users using the optional WITH GRANT OPTION clause.
- The user needs to have USAGE privilege on schema and appropriate privileges on the object.

When a user grants an explicit list of privileges, such as `GRANT INSERT, DELETE, REFERENCES ON applog TO Bob`:

- The GRANT statement succeeds only if all the roles are granted successfully. If any grant operation fails, the entire statement rolls back.
- Vertica will return ERROR if the user does not have grant options for the privileges listed.

When a user grants ALL privileges, such as `GRANT ALL ON applog TO Bob`, the statement always succeeds. Vertica grants all the privileges on which the grantor has the WITH GRANT OPTION and skips those privileges without the optional WITH GRANT OPTION.

For example, if the user Bob has delete privileges with the optional grant option on the applog table, only DELETE privileges are granted to Bob, and the statement succeeds:

```
=> GRANT DELETE ON applog TO Bob WITH GRANT OPTION;GRANT PRIVILEGE
```

For details, see the [GRANT Statements](#) in the SQL Reference Manual.

How to Revoke Privileges

In general, **ONLY** the user who originally granted a privilege can revoke it using a REVOKE statement. That user must have superuser privilege or have the optional WITH GRANT OPTION on the privilege. The user also must have USAGE privilege on the schema and appropriate privileges on the object for the REVOKE statement to succeed.

In order to revoke a privilege, this privilege must have been granted to the specified grantee by this grantor before. If Vertica finds that to be the case, the above REVOKE statement removes the privilege (and WITH GRANT OPTION privilege, if supplied) from the grantee. Otherwise, Vertica prints a NOTICE that the operation failed, as in the following example.

```
=> REVOKE SELECT ON applog FROM Bob;  
NOTICE 0: Cannot revoke "SELECT" privilege(s) for relation "applog" that you did not grant to "Bob"  
REVOKE PRIVILEGE
```

The above REVOKE statement removes the privilege (and WITH GRANT OPTION privilege, if applicable) from the grantee or it prints a notice that the operation failed.

In order to revoke grant option for a privilege, the grantor must have previously granted the grant option for the privilege to the specified grantee. Otherwise, Vertica prints a NOTICE.

The following REVOKE statement removes the GRANT option only but leaves the privilege intact:

```
=> GRANT INSERT on applog TO Bob WITH GRANT OPTION;  
GRANT PRIVILEGE  
=> REVOKE GRANT OPTION FOR INSERT ON applog FROM Bob;  
REVOKE PRIVILEGE
```

When a user revokes an explicit list of privileges, such as GRANT INSERT, DELETE, REFERENCES ON applog TO Bob:

- The REVOKE statement succeeds only if all the roles are revoked successfully. If any revoke operation fails, the entire statement rolls back.
- Vertica returns ERROR if the user does not have grant options for the privileges listed.
- Vertica returns NOTICE when revoking privileges that this user had not been previously granted.

When a user revokes ALL privileges, such as REVOKE ALL ON applog TO Bob, the statement always succeeds. Vertica revokes all the privileges on which the grantor has the optional WITH GRANT OPTION and skips those privileges without the WITH GRANT OPTION.

For example, if the user Bob has delete privileges with the optional grant option on the applog table, only grant option is revoked from Bob, and the statement succeeds without NOTICE:

```
=> REVOKE GRANT OPTION FOR DELETE ON applog FROM Bob;
```

For details, see the [REVOKE Statements](#) in the SQL Reference Manual.

Privilege Ownership Chains

The ability to revoke privileges on objects can cascade throughout an organization. If the grant option was revoked from a user, the privilege that this user granted to other users will also be revoked.

If a privilege was granted to a user or role by multiple grantors, to completely revoke this privilege from the grantee the privilege has to be revoked by each original grantor. The only exception is a superuser may revoke privileges granted by an object owner, with the reverse being true, as well.

In the following example, the SELECT privilege on table t1 is granted through a chain of users, from a superuser through User3.

- A superuser grants User1 CREATE privileges on the schema s1:

```
=> \c - dbadminYou are now connected as user "dbadmin".
=> CREATE USER User1;
CREATE USER
=> CREATE USER User2;
CREATE USER
=> CREATE USER User3;
CREATE USER
=> CREATE SCHEMA s1;
CREATE SCHEMA
=> GRANT USAGE on SCHEMA s1 TO User1, User2, User3;
GRANT PRIVILEGE
=> CREATE ROLE reviewer;
CREATE ROLE
=> GRANT CREATE ON SCHEMA s1 TO User1;
GRANT PRIVILEGE
```

- User1 creates new table t1 within schema s1 and then grants SELECT WITH GRANT OPTION privilege on s1.t1 to User2:

```
=> \c - User1You are now connected as user "User1".
=> CREATE TABLE s1.t1(id int, sourceID VARCHAR(8));
CREATE TABLE
=> GRANT SELECT on s1.t1 to User2 WITH GRANT OPTION;
GRANT PRIVILEGE
```

- User2 grants SELECT WITH GRANT OPTION privilege on s1.t1 to User3:

```
=> \c - User2You are now connected as user "User2".
=> GRANT SELECT ON s1.t1 to User3 WITH GRANT OPTION;
GRANT PRIVILEGE
```

- User3 grants SELECT privilege on s1.t1 to the reviewer role:

```
=> \c - User3You are now connected as user "User3".
=> GRANT SELECT ON s1.t1 to reviewer;
GRANT PRIVILEGE
```

Users cannot revoke privileges upstream in the chain. For example, User2 did not grant privileges on User1, so when User1 runs the following REVOKE command, Vertica rolls back the command:

```
=> \c - User2You are now connected as user "User2".
=> REVOKE CREATE ON SCHEMA s1 FROM User1;
ROLLBACK 0: "CREATE" privilege(s) for schema "s1" could not be revoked from "User1"
```

Users can revoke privileges indirectly from users who received privileges through a cascading chain, like the one shown in the example above. Here, users can use the CASCADE option to revoke privileges from all users "downstream" in the chain. A superuser or User1 can use the CASCADE option to revoke the SELECT privilege on table s1.t1 from all users. For example, a superuser or User1 can execute the following statement to revoke the SELECT privilege from all users and roles within the chain:

```
=> \c - User1You are now connected as user "User1".
=> REVOKE SELECT ON s1.t1 FROM User2 CASCADE;
REVOKE PRIVILEGE
```

When a superuser or User1 executes the above statement, the SELECT privilege on table s1.t1 is revoked from User2, User3, and the reviewer role. The GRANT privilege is also revoked from User2 and User3, which a superuser can verify by querying the [V_CATALOG.GRANTS](#) system table.

```
=> SELECT * FROM grants WHERE object_name = 's1' AND grantee ILIKE 'User%';
grantor | privileges_description | object_schema | object_name | grantee
-----+-----+-----+-----+-----
dbadmin | USAGE                  |               | s1          | User1
dbadmin | USAGE                  |               | s1          | User2
dbadmin | USAGE                  |               | s1          | User3
(3 rows)
```

Modifying Privileges

A superuser or object owner can use one of the ALTER statements to modify a privilege, such as changing a sequence owner or table owner. Reassignment to the new owner does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Changing Table Ownership

As a superuser or table owner, you can reassign table ownership with [ALTER TABLE](#), as follows:

```
ALTER TABLE [schema.]table-name OWNER TO owner-name
```

Changing table ownership is useful when moving a table from one schema to another. Ownership reassignment is also useful when a table owner leaves the company or changes job responsibilities. Because you can change the table owner, the tables won't have to be completely rewritten, you can avoid loss in productivity.

Changing table ownership automatically causes the following changes:

- Grants on the table that were made by the original owner are dropped and all existing privileges on the table are revoked from the previous owner. Changes in table ownership has no effect on schema privileges.
- Ownership of dependent IDENTITY/AUTO-INCREMENT sequences are transferred with the table. However, ownership does not change for independent sequences created with [CREATE SEQUENCE](#). To transfer ownership of these sequences, use [ALTER SEQUENCE](#).
- New table ownership is propagated to its projections.

Example

In this example, user Bob connects to the database, looks up the tables, and transfers ownership of table t33 from himself to user Alice.

```
=> \c - Bob
You are now connected as user "Bob".
=> \d
Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
public | applog | table | dbadmin |
public | t33 | table | Bob |
```

```
(2 rows)
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
```

Notice that when Bob looks up database tables again, he no longer sees table t33.

```
=> \d
                List of tables
                List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
(1 row)
```

When user Alice connects to the database and looks up tables, she sees she is the owner of table t33.

```
=> \c - Alice
You are now connected as user "Alice".
=> \d
                List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | t33 | table | Alice |
(2 rows)
```

Either Alice or a superuser can transfer table ownership back to Bob. In the following case a superuser performs the transfer.

```
=> \c - dbadmin
You are now connected as user "dbadmin".
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> \d
                List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
 public | comments | table | dbadmin |
 public | t33 | table | Bob |
 s1 | t1 | table | User1 |
(4 rows)
```

You can also query system table [V_CATALOG.TABLES](#) to view table and owner information. Note that a change in ownership does not change the table ID.

In the below series of commands, the superuser changes table ownership back to Alice and queries the TABLES system table.

```
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM tables;
table_schema_id | table_schema | table_id | table_name | owner_id | owner_name
-----+-----+-----+-----+-----+-----
```

```
45035996273704968 | public      | 45035996273713634 | applog   | 45035996273704962 | dbadmin
45035996273704968 | public      | 45035996273724496 | comments | 45035996273704962 | dbadmin
45035996273730528 | s1          | 45035996273730548 | t1       | 45035996273730516 | User1
45035996273704968 | public      | 45035996273795846 | t33      | 45035996273724576 | Alice
(5 rows)
```

Now the superuser changes table ownership back to Bob and queries the TABLES table again. Nothing changes but the owner_name row, from Alice to Bob.

```
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM tables;
  table_schema_id | table_schema |   table_id   | table_name |   owner_id   | owner_name--
-----+-----+-----+-----+-----+-----
45035996273704968 | public      | 45035996273713634 | applog     | 45035996273704962 | dbadmin
45035996273704968 | public      | 45035996273724496 | comments   | 45035996273704962 | dbadmin
45035996273730528 | s1          | 45035996273730548 | t1         | 45035996273730516 | User1
45035996273704968 | public      | 45035996273793876 | foo        | 45035996273724576 | Alice
45035996273704968 | public      | 45035996273795846 | t33        | 45035996273714428 | Bob
(5 rows)
```

See Also

[Changing Sequence Ownership](#)

Changing Sequence Ownership

The ALTER SEQUENCE command lets you change the attributes of an existing sequence. All changes take effect immediately, within the same session. Any parameters not set during an ALTER SEQUENCE statement retain their prior settings.

If you need to change sequence ownership, such as if an employee who owns a sequence leaves the company, you can do so with the following ALTER SEQUENCE syntax:

```
=> ALTER SEQUENCE sequence-name OWNER TO new-owner-name;
```

This operation immediately reassigns the sequence from the current owner to the specified new owner.

Only the sequence owner or a superuser can change ownership, and reassignment does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Note: Renaming a table owner transfers ownership of dependent sequence objects (associated IDENTITY/AUTO-INCREMENT sequences) but does not transfer ownership of other referenced sequences. See [Changing Table Ownership](#).

Example

The following example reassigns sequence ownership from the current owner to user Bob:

```
=> ALTER SEQUENCE sequential OWNER TO Bob;
```

See [ALTER SEQUENCE](#) in the SQL Reference Manual for details.

Viewing Privileges Granted on Objects

Vertica logs information about privileges granted on various objects, including the grantor and grantee, in the [V_CATALOG.GRANTS](#) system table. The order of columns in the table corresponds to the order in which they appear in the GRANT command. An asterisk in the output means the privilege was granted WITH GRANT OPTION.

The following command queries the GRANTS system table:

```
=> SELECT * FROM grants ORDER BY grantor, grantee;
```

grantor	privileges_description	object_schema	object_name	grantee
Bob			commentor	Alice
dbadmin	CREATE		schema2	Bob
dbadmin			commentor	Bob
dbadmin			commentor	Bob
dbadmin			logadmin	Bob
dbadmin	USAGE		general	Bob
dbadmin	INSERT, UPDATE, DELETE, REFERENCES	public	applog	Bob
dbadmin			logadmin	Ted
dbadmin	USAGE		general	Ted
dbadmin	USAGE		general	Sue
dbadmin	CREATE, CREATE TEMP		vmart	Sue
dbadmin	USAGE		public	Sue
dbadmin	SELECT*	public	applog	Sue
dbadmin	USAGE		general	Alice
dbadmin	INSERT, SELECT	public	comments	commentor
dbadmin	INSERT, SELECT	public	applog	commentor
dbadmin			logwriter	logadmin
dbadmin			logreader	logadmin
dbadmin	DELETE	public	applog	logadmin
dbadmin	SELECT	public	applog	logreader
dbadmin	INSERT	public	applog	logwriter
dbadmin	USAGE		v_internal	public
dbadmin	CREATE TEMP		vmart	public
dbadmin	USAGE		public	public
dbadmin	USAGE		v_catalog	public
dbadmin	USAGE		v_monitor	public
dbadmin	CREATE*, CREATE TEMP*		vmart	dbadmin
dbadmin	USAGE*, CREATE*		schema2	dbadmin
dbadmin	INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES*	public	comments	dbadmin
dbadmin	INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES*	public	applog	dbadmin

(30 rows)

To quickly find all of the privileges that have been granted to all users on the schema named myschema, run the following statement:

```
=> SELECT grantee, privileges_description FROM GRANTS WHERE object_name='myschema';
grantee | privileges_description
-----+-----
Bob     | USAGE, CREATE
Alice   | CREATE
(2 rows)
```

Note that the vsql commands, \dp and \z, both return similar information to GRANTS:

```
=> \dp
Access privileges for database "apps"
Privileges | Schema | Name | Grantee | Grantor |
-----+-----+-----+-----+-----+
public     | dbadmin | USAGE |         |         | v_internal
public     | dbadmin | USAGE |         |         | v_catalog
public     | dbadmin | USAGE |         |         | v_monitor
logadmin   | dbadmin |       |         |         | logreader
logadmin   | dbadmin |       |         |         | logwriter
Fred       | dbadmin | USAGE |         |         | general
Fred       | dbadmin |       |         |         | logadmin
Bob        | dbadmin | USAGE |         |         | general
dbadmin    | dbadmin | USAGE*, CREATE* |         |         | schema2
Bob        | dbadmin | CREATE |         |         | schema2
Sue        | dbadmin | USAGE |         |         | general
public     | dbadmin | USAGE |         |         | public
Sue        | dbadmin | USAGE |         |         | public
public     | dbadmin | CREATE TEMP |         |         | appdat
dbadmin    | dbadmin | CREATE*, CREATE TEMP* |         |         | appdat
Sue        | dbadmin | CREATE, CREATE TEMP |         |         | appdat
dbadmin    | dbadmin | INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES* | public | applog
logreader  | dbadmin | SELECT | public | applog
logwriter  | dbadmin | INSERT | public | applog
logadmin   | dbadmin | DELETE | public | applog
Sue        | dbadmin | SELECT* | public | applog
(22 rows)
```

See [GRANT Statements](#) in the SQL Reference Manual.

Access Policies

You can create the following access policy types to restrict access to sensitive information to only those users authorized to view it:

- [Column Access Policy](#)
- [Row Access Policy](#)

Important: If you have a table with both a row level access policy and a column level access policy, Vertica filters the row level access policy first. Then Vertica uses the column level access policy to filter the columns.

Use Cases

Column Access Policy Use Case

Base a column access policy on a user's role and the privileges granted to that role.

For example, in a healthcare organization, customer support representatives and account managers have access to the same customer table. The table contains the column SSN for storing customer Social Security numbers. Customer support representatives have only partial access, to view the last four digits. The account manager, however, must be able to view entire Social Security numbers. Therefore, the *manager* role has privileges to view all nine digits of the social security numbers.

When creating a column access policy, use expressions to specify exactly what different users or roles can access within the column.

In this case, a *manager* can access the entire SSN column, while customer support representatives can only access the last four digits:

```
=> CREATE ACCESS POLICY ON schema.customers_table
FOR COLUMN SSN
CASE
WHEN ENABLED_ROLE('manager') THEN SSN
else substr(SSN, 8, 4)
END
ENABLE;
```

Row Access Policy Use Case

You can also create a row access policy on the same table. For example, you can modify access to a customer table so a manager can view data in all rows. However, a broker can see a row only if the customer is associated with that broker:

```
=> select * from customers_table;
```

custID	password	ssn
1	secret	12345678901
2	secret	12345678902
3	secret	12345678903

(3 rows)

Each customer in the `customers_table` has an assigned broker:

```
=> select * from broker_info;
```

broker	custID
u1	1
u2	2
u3	3

Create the access policy to allow a *manager* to see all data in all rows. Limit a broker's view to only those customers to which the broker is assigned:

```
=> CREATE ACCESS POLICY ON schema.customers_table
  FOR rows
  WHERE
    ENABLED_ROLE('manager')
    or
    (ENABLED_ROLE('broker') AND customers_table.custID in (SELECT broker_info.custID FROM broker_
info
  WHERE broker = CURRENT_USER()))
  ENABLE;
```

Access Policy Creation Workflow

You can create access policies for any table type, columnar, external, or [flex](#). You can also create access policies on any column type, including joins.

If no users or roles are already created, you must create them before creating an access policy:

- [Create a User](#)
- [Create a Role](#)
- [GRANT \(Schema\)](#)
- [GRANT \(Table\)](#)
- [Grant a user access to the role](#)
- The user [enables the role](#) with the [SET ROLE](#) statement (unless the administration user assigned a [default role](#) to the user)
- Create the access policy with the [CREATE ACCESS POLICY](#) statement.

Working With Access Policies

This section describes areas that may affect how you use access policies.

Performing Operations

Having row and column access policies enabled on a table may affect the behavior when you attempt to perform the following DML operations:

- Insert
- Update
- Delete
- Merge
- Copy
- Select

Row Level Access Behavior

On tables where a row access policy is enabled, you can only perform DML operations when the condition in the Row access policy evaluates to TRUE. For example:

Table1 appears as follows:

```
A | B
---+---
1 | 1
2 | 2
3 | 3
```

Create the following row access policy on Table1:

```
=> CREATE ACCESS POLICY on table1 for ROWS
WHERE enabled_role('manager')
OR
A<2
ENABLE;
```

With this policy enabled, the following behavior exists for users who want to perform DML operations:

- A user with the manager role can perform DML on all rows in the table, because the WHERE clause in the policy evaluates to TRUE.
- Users with non-manager roles can only perform a SELECT to return data in column A that has a value of less than two. If the access policy has to read the data in the table to confirm a condition, it does not allow DML operations.

Column Level Access Behavior

On tables where a column access policy is enabled, you can perform DML operations if you can view the entire column. For example:

Table1 appears as follows:

```
A | B
---+---
1 | 1
2 | 2
3 | 3
```

Create the following column access policy on Table1:

```
=> CREATE ACCESS POLICY on Table1 FOR column A NULL::int enable;
```

In this case users cannot perform DML operations on column A.

Important: Users who can access all the rows and columns in a table with an access policy enabled can perform DML operations. Therefore, when you create an access policy, make sure you construct it in a manner that all row and column data is accessible by at least one user. This allows at least one user to perform any DML that may be required. Otherwise, you can temporarily disable the access policy to perform DML.

Schema Table and Privileges

Only dbadmin users can create access policies. If you want a user to be able to use access policies, you must first assign that user the appropriate privileges.

- Grant [schema](#) or [table](#) privileges to a table non-owner to allow that user to use the access policy.
- Revoke [schema](#) or [table](#) privileges to prohibit the user from using the access policy.

This example shows how you can create an access policy without the user being granted privilege for the public schema:

```
=> CREATE ACCESS POLICY ON public.customers_table  
FOR COLUMN SSN  
WHEN ENABLED_ROLE('operator') THEN SUBSTR(SSN, 8, 4)
```

Enable and Disable Access Policy Creation

Access policies are enabled by default for all tables in the database. To disable and enable the creation of new access policies at the database level, use the ALTER DATABASE statement.

Disable Creation of New Access Policies

```
=> ALTER DATABASE dbname SET EnableAccessPolicy=0;
```

Enable Creation of New Access Policies

```
=> ALTER DATABASE dbname SET EnableAccessPolicy=1;
```

Limitations on Creating Access Policies with Projections

You can create access policies on columns in tables that are part of a projection. However, you cannot create an access policy on an *input table* for the following projections:

- Top-K projections
- Aggregate projections
- Projections with expressions

Sometimes, a table already has an access policy and is part of a projection. In such cases, if the Vertica optimizer cannot fold (or *compress*) the query, the access query is blocked.

Query Optimization Considerations

When using access policies be aware of the following potential behaviors, and design tables optimally.

Design Tables That All Authorized Users Can Access

When Database Designer creates projections for a given table, it takes into account the access policies that apply to the current user. The set of projections that Database Designer produces for the table are optimized for that user's access privileges, and other users with similar access privileges. However, these projections might be less than optimal for users with different

access privileges. These differences might have some effect on how efficiently Vertica processes queries from those users. Therefore, when you evaluate projection designs for that table using Database Designer, design a given table so that all authorized users have optimal access.

Avoid Performance Issues Caused by Dynamic Rewrite

To enforce row-level access policies, the system dynamically rewrites user queries. Therefore, query performance may be affected by how row-level access policies are written.

For example, referring to preceding [Access Policy Use Cases](#), run the following query. Enable both the row and column access policies on the `customers_table`:

```
=> SELECT * from customers_table;
```

Vertica rewrites this query plan to:

```
=> SELECT * from (select custID, password, CASE WHEN enabled_role('manager') THEN SSN else substr
(SSN, 8, 4) end AS SSN
FROM customers_table
WHERE
enabled_role('broker')
AND
customers_table.custID IN (SELECT broker_info.custID FROM broker_info WHERE broker = current_user())
) customers_table;
```

Column Access Policy

Use the [CREATE ACCESS POLICY](#) statement to create a column access policy for a specific column in a table. Creating an access policy depends on the expressions specified when creating the policy, and also on the following:

- [Viewing a User's Role](#)
- [Granting Privileges to Roles](#)

Example

Suppose you want to prevent users from viewing a specific column in a table. This example shows how to create an access policy that masks column A in Table1.

Run the following SQL command:

```
=> SELECT * FROM Table1;
```

Table1 appears as follows:

```
A | B
---+-----
1 | one
2 | two
3 | three
4 | four
```

Create the following column access policy:

```
=> CREATE ACCESS POLICY on Table1 FOR column A NULL::int enable;
```

Re-run the SQL command:

```
=> SELECT * FROM Table1;
```

The following is returned:

```
A | B
---+-----
  | one
  | two
  | three
  | four
```

Note that no values appear in column A because the access policy prevents the return of this data (NULL::int).

Creating Column Access Policies

Creating a column access policy allows different users to run the same query and receive different results. For example, you can create an access policy authorizing access to a column of bank account numbers. You can specify that a user with the role *employee* cannot access this information. However, you do give access to a user with a *manager* role.

Conditions specified in the access policy determine whether the user can see data restricted by the policy. This example shows how you can specify that the *manager* role can view the entire Social Security number while the *operator* role can only view the last four digits. The first five digits are masked for the operator role (THEN SUBSTR (SSN, 8, 4)). The 8 indicates the operator sees data starting on the eighth character (such as 123-45-6789).

```
=> CREATE ACCESS POLICY ON customers_table
FOR COLUMN SSN
```

```
CASE  
WHEN ENABLED_ROLE('manager') THEN SSN  
WHEN ENABLED_ROLE('operator') THEN SUBSTR(SSN, 8, 4)  
ELSE NULL  
END  
ENABLE;
```

Access Policy Limitations

When you use column access policies, be aware of the following limitations:

- When using an access policy you cannot use any of the following in an expression:
 - Aggregate functions
 - Subquery
 - Analytics
 - UDT
- If the query cannot be folded by the Vertica optimizer, all functions other than SELECT are blocked. The following error message appears:

```
ERROR 0: Unable to INSERT: "Access denied due to active access policy on table <tablename> for column <columnname>
```

Note: Folding a query refers to the act of replacing deterministic expressions involving only constants, with their computed values.

- You cannot create a column access policy on temporary tables.
- It is recommended to not use a column access policy on a flex table. If you create a column access policy on a flex table, the following appears:

```
WARNING 0: Column Access Policies on flex tables may not be completely secure
```

Examples

The following examples show how to create a column access policy for various situations.

Create Access Policy in Public Schema for Column in Customer Table

```
=> CREATE ACCESS POLICY on public.customer FOR COLUMN cid length('xxxxx') enable;
```

Use Expression to Further Specify Data Access and Restrictions

In this example, a user with a *supervisor* role can see data from the `deal_size` column in the `vendor_dimension` table. However, a user assigned an *employee* role cannot.

```
=> CREATE ACCESS POLICY ON vendor_dimension FOR COLUMN deal_size  
CASE  
WHEN ENABLED_ROLE('supervisor') THEN deal_size  
WHEN ENABLED_ROLE('employee') THEN NULL  
END  
ENABLE;
```

Substitute Specific Data for Actual Data in Column

In this example, the value 1000 appears rather than the actual column data:

```
=> CREATE ACCESS POLICY on public.customer FOR COLUMN cid 1000 enable;  
  
=> SELECT * FROM customer;  
cid | dist_code  
-----+-----  
1000 | 2  
1000 | 10  
(2 rows)
```

See Also

- [CREATE ACCESS POLICY](#)
- [ALTER ACCESS POLICY](#)
- [DROP ACCESS POLICY](#)

Enable or Disable Column Access Policy

If you have dbadmin privileges, you can enable and disable an individual access policy in a table, as the following examples show.

Enable Column Access Policy

```
=> ALTER ACCESS POLICY on customer FOR column customer_key enable;
```

Disable Column Access Policy

```
=> ALTER ACCESS POLICY on customer FOR column customer_key disable;
```

Row Access Policy

Use the [CREATE ACCESS POLICY](#) statement to create a row access policy for a specific row in a table. You must use a WHERE clause to set the access policy's condition.

Example

Suppose you have a customers table but want to limit users with the broker role to being able to view only customers to which they are assigned. This example shows how to create a row access policy that allows managers to view everything and brokers to view only their customers.

Run the following SQL statement:

```
=> SELECT * FROM customers_table;
```

The customers_table appears as follows:

custID	password	SSN
1	secret	123456789

```
 2 | secret | 123456780  
 3 | secret | 123456781  
(3 rows)
```

Run the following SQL statement:

```
=> SELECT * FROM broker_info;
```

The `broker_info` table shows that each customer has an assigned broker:

```
broker | custID  
-----+-----  
 user1 |      1  
 user2 |      2  
 user3 |      3  
(3 rows)
```

Create the following access policy that only allows brokers to see customers to which they are associated:

```
=> CREATE ACCESS POLICY on customers_table for rows  
WHERE  
  ENABLED_ROLE('manager')  
  or  
  (ENABLED_ROLE('broker') AND customers_table.custID in (SELECT broker_info.custID FROM broker_info  
  WHERE broker = CURRENT_USER()))  
ENABLE;
```

As `user1`, run the following SQL command:

```
user1=> SELECT * FROM customers_table;
```

The following is returned because `user1` is associated with `custID 1`:

```
custID | password | SSN  
-----+-----+-----  
 1 | secret | 123456789  
(1 rows)
```

Creating Row Access Policies

Creating a row access policy determines what rows a user can access during a query. Row access policies include a `WHERE` clause that prompts the query to return only those rows where the condition is true. For example, a user with a `BROKER` role should only be able to

access customer information for which the user is a broker. You can write a predicate for this situation as follows:

```
WHERE ENABLED_ROLE('broker') AND customers_table.custID in (SELECT broker_info.custID FROM broker_info WHERE broker = CURRENT_USER())
```

You can use a row access policy to enforce this restriction. The following example shows how you can create a row access policy. This policy limits a user with a broker role to access information for customers whose custID in the customers_table matches the custID in the broker_info table.

```
=> CREATE ACCESS POLICY on customers_table  
for rows  
WHERE  
    ENABLED_ROLE('broker')  
    AND  
    customers_table.custID in (SELECT broker_info.custID FROM broker_info WHERE broker = CURRENT_USER  
(  
))  
enable;
```

Row Access Policy Limitations

Be aware of the following limitations when using row access policies:

- You can only have one row access policy per table. If you need to add more later, place the policies in a single `WHERE` predicate and use [ALTER ACCESS POLICY](#) to enable the new condition.
- You cannot use row access policies on:
 - Tables with aggregate projections
 - Temporary tables
 - System tables. If you try to create a row access policy on a system table the following message appears:

```
=> ROLLBACK 0: Access policy cannot be created on system table <system table name>
```

- Views
- When a row access policy exists on a table, you cannot create directed queries on that table.

Examples

The following examples show you can create a row access policy:

Create Access Policy in for specific row in Customer Table

```
=> CREATE ACCESS POLICY on customer FOR ROWS where cust_id > 3 enable;
```

See Also

- [CREATE ACCESS POLICY](#)
- [ALTER ACCESS POLICY](#)
- [DROP ACCESS POLICY](#)

Enable or Disable Row Access Policy

If you have dbadmin privileges, you can enable and disable individual row access policies in a table, as the following examples show:

Enable Row Access Policy

```
=> ALTER ACCESS POLICY on customer FOR rows enable;
```

Disable Row Access Policy

```
=> ALTER ACCESS POLICY on customer FOR rows disable;
```

About Database Roles

To make managing permissions easier, use roles. A role is a collection of privileges that a superuser can grant to (or revoke from) one or more users or other roles. Using roles avoids having to manually grant sets of privileges user by user. For example, several users might be assigned to the administrator role. You can grant or revoke privileges to or from the administrator role, and all users with access to that role are affected by the change.

Note: Users must first enable a role before they gain all of the privileges that have been granted to it. See [Enabling Roles](#).

Role Hierarchies

You can also use roles to build hierarchies of roles; for example, you can create an administrator role that has privileges granted non-administrator roles as well as to the privileges granted directly to the administrator role. See also [Role Hierarchy](#).

Roles do not supersede manually-granted privileges, so privileges directly assigned to a user are not altered by roles. Roles just give additional privileges to the user.

Creating and Using a Role

Using a role follows this general flow:

1. A superuser creates a role using the [CREATE ROLE](#) statement.
2. A superuser or object owner grants privileges to the role using one of the GRANT statements.
3. A superuser or users with administrator access to the role grant users and other roles access to the role.
4. Users granted access to the role use the [SET ROLE](#) command to enable that role and gain the role's privileges.

You can do steps 2 and 3 in any order. However, granting access to a role means little until the role has privileges granted to it.

Tip: You can query the V_CATALOG system tables [ROLES](#), [GRANTS](#), and [USERS](#) to see any directly-assigned roles; however, these tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly). See the [HAS_ROLE\(\)](#) function for additional information.

Roles on Management Console

When users sign in to the Management Console (MC), what they can view or do is governed by MC roles. For details, see [About MC Users](#) and [About MC Privileges and Roles](#).

Types of Database Roles

Vertica has the following pre-defined roles:

- [PUBLIC](#)
- [PSEUDOSUPERUSER](#)
- [DBADMIN](#)
- [DBDUSER](#)
- [SYSMONITOR](#)

You cannot drop or rename predefined roles, however, you can grant to, or revoke from, predefined roles except to/from PUBLIC. You can also grant predefined roles to other roles and users.

Individual privileges may be granted to/revoked from predefined roles. See the SQL Reference Manual for all of the GRANT and REVOKE statements.

DBADMIN Role

Every database has the special DBADMIN role. A superuser (or someone with the [PSEUDOSUPERUSER Role](#)) can grant this role to or revoke this role from any user or role.

Users who enable the DBADMIN role gain these privileges:

- Create or drop users
- Create or drop schemas
- Create or drop roles
- Grant roles to other users
- View all system tables
- View and terminate user sessions
- Access to all data created by any user

The DBADMIN role does NOT allow users to:

- Start and stop a database
- Set configuration parameters

Note: A user with a DBADMIN role must have the ADMIN OPTION enabled to be able to grant a DBADMIN or PSEUDOSUPERUSER role to another user. . For more information see [GRANT \(Role\)](#).

You cannot assign additional roles to the DBADMIN role:

```
=> CREATE ROLE appviewer;  
CREATE ROLE  
=> GRANT appviewer TO dbadmin;  
ROLLBACK 2347: Cannot alter predefined role "dbadmin"
```

You can, however, grant the DBADMIN role to other roles to augment a set of privileges. See [Role Hierarchy](#) for more information.

View a List of Database Superusers

To see who is a superuser, run the `vsq! \du` command. In this example, only `dbadmin` is a superuser.

```
=> \du  
List of users  
User name | Is Superuser  
-----+-----  
dbadmin   | t  
Fred      | f  
Bob       | f  
Sue       | f  
Alice     | f  
User1     | f  
User2     | f  
User3     | f  
u1        | f  
u2        | f  
(10 rows)
```

See Also

[Database Administration User](#)

DBDUSER Role

The DBDUSER role is predefined, and allows non-DBADMIN users to access Database Designer [functions](#). This role must be explicitly granted by a superuser or DBADMIN user.

Note: Non-DBADMIN users with the DBDUSER role cannot run Database Designer through Administration Tools. Only [DBADMIN](#) users can run Administration Tools.

You cannot assign any additional privileges to the DBDUSER role, but you can grant the DBDUSER role to other roles to augment a set of privileges.

After you are granted the DBDUSER role, you must enable it before you can run Database Designer using command-line functions. For more information, see [About Running Database Designer Programmatically](#).

Important: When you create a DBADMIN user or grant the DBDUSER role, make sure to associate a resource pool with that user to manage resources during Database Designer runs. Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. When a user runs Database Designer, either using the Administration Tools or programmatically, its execution is mostly contained by the user's resource pool, but may spill over into some system resource pools for less-intensive tasks.

PSEUDOSUPERUSER Role

The special PSEUDOSUPERUSER role is automatically created in each database. A superuser (or someone with the PSEUDOSUPERUSER role) can perform grant and revoke on this role. The PSEUDOSUPERUSER cannot revoke or change any superuser privileges.

Users with the PSEUDOSUPERUSER role are entitled to complete administrative privileges, including the ability to:

- Create schemas
- Create and grant privileges to roles
- Bypass all GRANT/REVOKE authorization
- Set user account's passwords
- Lock and unlock user accounts

- Create or drop a UDF library
- Create or drop a UDF function
- Create or drop an external procedure
- Add or edit comments on nodes
- Create or drop password profiles

You cannot revoke any of these privileges from a PSEUDOSUPERUSER.

You can assign additional privileges to the PSEUDOSUPERUSER role, but you cannot assign any additional roles; for example, the following is not allowed:

```
=> CREATE ROLE appviewer;  
CREATE ROLE  
=> GRANT appviewer TO pseudosuperuser;  
ROLLBACK 2347: Cannot alter predefined role "pseudosuperuser"
```

PUBLIC Role

By default, every database has the special PUBLIC role. Vertica grants this role to each user automatically, and it is automatically enabled. You grant privileges to this role that every user should have by default. You can also grant access to roles to PUBLIC, which allows any user to access the role using the [SET ROLE](#) statement.

Note: The PUBLIC role can never be dropped, nor can it be revoked from users or roles.

Privileges created using the WITH GRANT OPTION cannot be granted to a Public Role:

```
=> CREATE TABLE t1(a int);  
CREATE TABLE  
=> GRANT SELECT on t1 to PUBLIC with grant option;  
ROLLBACK 3484: Grant option for a privilege cannot be granted to  
"public"
```

For more information see [How to Grant Privileges](#).

Example

In the following example, if the superuser hadn't granted INSERT privileges on the table publicdata to the PUBLIC group, the INSERT statement executed by user bob would fail:

```
=> CREATE TABLE publicdata (a INT, b VARCHAR);
CREATE TABLE
=> GRANT INSERT, SELECT ON publicdata TO PUBLIC;
GRANT PRIVILEGE
=> CREATE PROJECTION publicdataproj AS (SELECT * FROM publicdata);
CREATE PROJECTION

dbadmin=> \c - bob
You are now connected as user "bob".

=> INSERT INTO publicdata VALUES (10, 'Hello World');
OUTPUT
-----
      1
(1 row)
```

See Also

[PUBLIC User](#)

SYSMONITOR Role

An organization's database administrator may have many responsibilities outside of maintaining Vertica as a DBADMIN user. In this case, as the DBADMIN you may want to delegate some Vertica administrative tasks to another Vertica user.

The DBADMIN can assign a delegate the SYSMONITOR role to grant access to system tables without granting full [DBADMIN](#) access.

The SYSMONITOR role provides the following privileges.

- View all system tables that are marked as monitorable. You can see a list of all the monitorable tables by issuing the statement:

```
=> select * from system_tables where is_monitorable='t';
```

- If `WITH ADMIN OPTION` was included when granting SYSMONITOR to the user or role, that user or role can then grant SYSMONITOR privileges to other users and roles.

Grant a SYSMONITOR Role

To grant a user or role the SYSMONITOR role, you must be one of the following:

- a DBADMIN user
- a user assigned the SYSMONITOR who has the ADMIN OPTION

Use the [GRANT \(Role\)](#) SQL statement to assign a user the SYSMONITOR role. This example shows how to grant the SYSMONITOR role to user1 and includes administration privileges by using the WITH ADMIN OPTION parameter. The ADMIN OPTION grants the SYSMONITOR role administrative privileges.

```
=> GRANT SYSMONITOR TO user1 WITH ADMIN OPTION;
```

This example shows how to revoke the ADMIN OPTION from the SYSMONITOR role for user1:

```
=> REVOKE ADMIN OPTION for SYSMONITOR FROM user1;
```

Use CASCADE to revoke ADMIN OPTION privileges for all users assigned the SYSMONITOR role:

```
=> REVOKE ADMIN OPTION for SYSMONITOR FROM PUBLIC CASCADE;
```

Example

This example shows how to:

- Create a user
- Create a role
- Grant SYSMONITOR privileges to the new role
- Grant the role to the user

```
=> CREATE USER user1;  
=> CREATE ROLE monitor;  
=> GRANT SYSMONITOR to monitor;  
=> GRANT monitor to user1;
```

Assign SYSMONITOR Privileges

This example uses the user and role created in the Grant SYSMONITOR Role example and shows how to:

- Create a table called `personal_data`
- Log in as `user1`
- Grant `user1` the monitor role. (You already granted the monitor `SYSMONITOR` privileges in the Grant a `SYSMONITOR` Role example.)
- Run a `SELECT` statement as `user1`

The results of the operations are based on the privilege already granted to `user1`.

```
=> CREATE TABLE personal_data (SSN varchar (256));
=> \c -user1;
user1=> SET ROLE monitor;
user1=> SELECT COUNT(*) FROM TABLES;
COUNT
-----
1
(1 row)
```

Because you assigned the `SYSMONITOR` role, `user1` can see the number of rows in the `Tables` system table. In this simple example, there is only one table (`personal_data`) in the database so the `SELECT COUNT` returns one row. In actual conditions, the `SYSMONITOR` role would see all the tables in the database.

Check if a Table is Accessible by `SYSMONITOR`

Use the following command to check if a system table can be accessed by a user assigned the `SYSMONITOR` role:

```
=> select table_name, is_monitorable from system_tables where table_
name='<table_name>';
```

Example

This example checks whether the `current_session` system table is accessible by the `SYSMONITOR`:

```
=> select table_name, is_monitorable from system_tables where table_
name='current_session';
table_name      | is_monitorable
-----
current_session | t
```

The `t` in the `is_monitorable` column indicates the `current_session` system table is accessible by the `SYSMONITOR`.

Default Roles for Database Users

By default, no roles (other than the default [PUBLIC Role](#)) are enabled at the start of a user session.

```
=> SHOW ENABLED_ROLES;
  name      | setting
-----+-----
enabled roles |
(1 row)
```

A superuser can set one or more default roles for a user, which are automatically enabled at the start of the user's session. Setting a default role is a good idea if users normally rely on the privileges granted by one or more roles to carry out the majority of their tasks. To set a default role, use the `DEFAULT ROLE` parameter of the [ALTER USER](#) statement as superuser:

```
=> \c vmart apps
You are now connected to database "apps" as user "dbadmin".
=> ALTER USER Bob DEFAULT ROLE logadmin;
ALTER USER
=> \c - Bob;
You are now connected as user "Bob"
=> SHOW ENABLED_ROLES;
  name      | setting
-----+-----
enabled roles | logadmin
(1 row)
```

Notes

- Only roles that the user already has access to can be made default.
- Unlike granting a role, setting a default role or roles overwrites any previously-set defaults.
- To clear any default roles for a user, use the keyword `NONE` as the role name in the `DEFAULT ROLE` argument.
- Default roles only take effect at the start of a user session. They do not affect the roles enabled in the user's current session.
- Avoid giving users default roles that have administrative or destructive privileges (the [PSEUDOSUPERUSER](#) role or `DROP` privileges, for example). By forcing users to explicitly enable these privileges, you can help prevent accidental data loss.

Using Database Roles

There are several steps to using roles:

1. A superuser creates a role using the [CREATE ROLE](#) statement.
2. A superuser or object owner grants privileges to the role.
3. A superuser or users with administrator access to the role grant users and other roles access to the role.
4. Users granted access to the role run the [SET ROLE](#) command to make that role active and gain the role's privileges.

You can do steps 2 and 3 in any order. However, granting access to a role means little until the role has privileges granted to it.

Tip: Query system tables [ROLES](#), [GRANTS](#), and [USERS](#) to see any directly-assigned roles. Because these tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly), see the [HAS_ROLE\(\)](#) function for additional information.

See Also

- [About MC Privileges and Roles](#)

Role Hierarchy

In addition to granting roles to users, you can also grant roles to other roles. This lets you build hierarchies of roles, with more privileged roles (an administrator, for example) being assigned all of the privileges of lesser-privileged roles (a user of a particular application), in addition to the privileges you assign to it directly. By organizing your roles this way, any privilege you add to the application role (reading or writing to a new table, for example) is automatically made available to the more-privileged administrator role.

Example

The following example creates two roles, assigns them privileges, then assigns them to a new administrative role.

1. Create new table applog:

```
=> CREATE TABLE applog (id int, sourceID VARCHAR(32), data TIMESTAMP, event VARCHAR(256));
```

2. Create a new role called logreader:

```
=> CREATE ROLE logreader;
```

3. Grant the logreader role read-only access on the applog table:

```
=> GRANT SELECT ON applog TO logreader;
```

4. Create a new role called logwriter:

```
=> CREATE ROLE logwriter;
```

5. Grant the logwriter write access on the applog table:

```
=> GRANT INSERT ON applog to logwriter;
```

6. Create a new role called logadmin, which will rule the other two roles:

```
=> CREATE ROLE logadmin;
```

7. Grant the logadmin role privileges to delete data:

```
=> GRANT DELETE ON applog to logadmin;
```

8. Grant the logadmin role privileges to have the same privileges as the logreader and logwriter roles:

```
=> GRANT logreader, logwriter TO logadmin;
```

9. Create new user Bob:

```
=> CREATE USER Bob;
```

10. Give Bob logadmin privileges:

```
=> GRANT logadmin TO Bob;
```

The user Bob can now enable the logadmin role, which also includes the logreader and logwriter roles. Note that Bob cannot enable either the logreader or logwriter role directly. A user can only enable explicitly-granted roles.

Hierarchical roles also works with administrative access to a role:

```
=> GRANT logreader, logwriter TO logadmin WITH ADMIN OPTION;  
GRANT ROLE  
=> GRANT logadmin TO Bob;  
=> \c - bob; -- connect as Bob  
You are now connected as user "Bob".  
=> SET ROLE logadmin; -- Enable logadmin role  
SET  
=> GRANT logreader TO Alice;  
GRANT ROLE
```

Note that the user Bob only has administrative access to the logreader and logwriter roles through the logadmin role. He doesn't have administrative access to the logadmin role, since it wasn't granted to him with the optional WITH ADMIN OPTION argument:

```
=> GRANT logadmin TO Alice;  
WARNING: Some roles were not granted  
GRANT ROLE
```

For Bob to be able to grant the logadmin role, a superuser would have had to explicitly grant him administrative access.

See Also

- [About MC Privileges and Roles](#)

Creating Database Roles

A superuser creates a new role using the [CREATE ROLE](#) statement. Only a superuser can create or drop roles.

```
=> CREATE ROLE administrator;  
CREATE ROLE
```

The newly-created role has no privileges assigned to it, and no users or other roles are initially granted access to it. A superuser must [grant privileges](#) and [access](#) to the role.

Deleting Database Roles

A superuser can delete a role with the [DROP ROLE](#) statement.

Note that if any user or other role has been assigned the role you are trying to delete, the `DROP ROLE` statement fails with a dependency message.

```
=> DROP ROLE administrator;
NOTICE:  User Bob depends on Role administrator
ROLLBACK:  DROP ROLE failed due to dependencies
DETAIL:   Cannot drop Role administrator because other objects depend on it
HINT:     Use DROP ROLE ... CASCADE to remove granted roles from the dependent users/roles
```

Supply the optional `CASCADE` parameter to drop the role and its dependencies.

```
=> DROP ROLE administrator CASCADE;
DROP ROLE
```

Granting Privileges to Roles

A superuser or owner of a schema, table, or other database object can assign privileges to a role, just as they would assign privileges to an individual user by using the `GRANT` statements described in the [SQL Reference Manual](#). See [About Database Privileges](#) for information about which privileges can be granted.

Granting a privilege to a role immediately affects active user sessions. When you grant a new privilege, it becomes immediately available to every user with the role active.

Example

The following example creates two roles and assigns them different privileges on a single table called `applog`.

1. Create a table called `applog`:

```
=> CREATE TABLE applog (id int, sourceID VARCHAR(32), data TIMESTAMP, event VARCHAR(256));
```

2. Create a new role called `logreader`:

```
=> CREATE ROLE logreader;
```

3. Assign read-only privileges to the logreader role on table applog:

```
=> GRANT SELECT ON applog TO logreader;
```

4. Create a role called logwriter:

```
=> CREATE ROLE logwriter;
```

5. Assign write privileges to the logwriter role on table applog:

```
=> GRANT INSERT ON applog TO logwriter;
```

See the [SQL Reference Manual](#) for the different GRANT statements.

Revoking Privileges From Roles

Use one of the REVOKE statements to revoke a privilege from a role.

```
=> REVOKE INSERT ON applog FROM logwriter;  
REVOKE PRIVILEGE
```

Revoking a privilege immediately affects any user sessions that have the role active. When you revoke a privilege, it is immediately removed from users that rely on the role for the privilege.

See the [SQL Reference Manual](#) for the different REVOKE statements.

Granting Access to Database Roles

A pseudosuperuser or dbadmin user can assign any role to a user or to another role using the GRANT command. The simplest form of this command is:

```
GRANT role [, ...] TO { user | role } [, ...]
```

Vertica returns a NOTICE if you grant a role to a user who has already been granted that role. For example:

```
=> GRANT commenter to Bob;  
NOTICE 4622: Role "commenter" was already granted to user "Bob"
```

See [GRANT \(Role\)](#) in the SQL Reference Manual for details.

Example

The following example shows how to create a role called `commenter` and grant that role to user Bob:

1. Create a table called `comments`.

```
=> CREATE TABLE comments (id INT, comment VARCHAR);
```

2. Create a role called `commenter`.

```
=> CREATE ROLE commenter;
```

3. Grant privileges to the `commenter` role on the `comments` table.

```
=> GRANT INSERT, SELECT ON comments TO commenter;
```

4. Grant the `commenter` role to user Bob.

```
=> GRANT commenter TO Bob;
```

Before being able to access the role and its associated privileges, Bob must enable the newly-granted role to himself.

1. Connect to the database as user Bob.

```
=> \c - Bob
```

2. Enable the role.

```
=> SET ROLE commenter;
```

3. Insert some values into the `comments` table.

```
=> INSERT INTO comments VALUES (1, 'Hello World');
```

Based on the privileges granted to Bob by the `commenter` role, Bob can insert and query the `comments` table.

4. Query the `comments` table.

```
=> SELECT * FROM comments;
 id | comment
-----+-----
  1 | Hello World
(1 row)
```

5. Commit the transaction.

```
=> COMMIT;
```

Note that Bob does not have proper permissions to drop the table.

```
=> DROP TABLE comments;ROLLBACK 4000: Must be owner of relation comments
```

See Also

- [Granting Database Access to MC Users](#)

Revoking Access From Database Roles

A superuser can revoke any role from a user or from another role using the REVOKE command. The simplest form of this command is:

```
REVOKE role [, ...] FROM { user | role | PUBLIC } [, ...]
```

See [REVOKE \(Role\)](#) in the SQL Reference Manual for details.

Example

To revoke access from a role, use the REVOKE (Role) statement:

1. Connect to the database as a superuser:

```
\c - dbadmin
```

2. Revoke the commenter role from user Bob:

```
=> REVOKE commenter FROM bob;
```

Granting Administrative Access to a Role

A superuser can assign a user or role administrative access to a role by supplying the optional `WITH ADMIN OPTION` argument to the `GRANT` statement. Administrative access allows the user to grant and revoke access to the role for other users (including granting them administrative access). Giving users the ability to grant roles lets a superuser delegate role administration to other users.

Example

The following example demonstrates granting the user bob administrative access to the commenter role, then connecting as bob and granting a role to another user.

1. Connect to the database as a superuser (or a user with administrative access):

```
=> \c - dbadmin
```

2. Grant administrative options on the commenter role to Bob

```
=> GRANT commenter TO Bob WITH ADMIN OPTION;
```

3. Connect to the database as user Bob

```
=> \c - Bob
```

4. As user Bob, grant the commenter role to Alice:

```
=> GRANT commenter TO Alice;
```

Users with administrative access to a role can also grant other users administrative access:

```
=> GRANT commenter TO alice WITH ADMIN OPTION;  
GRANT ROLE
```

As with all user privilege models, database superusers should be cautious when granting any user a role with administrative privileges. For example, if the database superuser grants two users a role with administrative privileges, both users can revoke the role of the other user. This example shows granting the `appadmin` role (with administrative privileges) to users `bob` and `alice`. After each user has been granted the `appadmin` role, either user can connect as the other will full privileges.

```
=> GRANT appadmin TO bob, alice WITH ADMIN OPTION;  
GRANT ROLE  
=> \connect - bob  
You are now connected as user "bob".  
=> REVOKE appadmin FROM alice;  
REVOKE ROLE
```

Revoking Administrative Access From a Role

A superuser can revoke administrative access from a role using the `ADMIN OPTION` parameter with the `REVOKE` statement. Giving users the ability to revoke roles lets a superuser delegate role administration to other users.

Example

The following example demonstrates revoking administrative access from Alice for the commenter role.

1. Connect to the database as a superuser (or a user with administrative access)

```
\c - dbadmin
```

2. Issue the `REVOKE` command with `ADMIN OPTION` parameters:

```
=> REVOKE ADMIN OPTION FOR commenter FROM alice;
```

Enabling Roles

By default, roles aren't enabled automatically for a user account. (See [Default Roles for Database Users](#) for a way to make roles enabled automatically.) Users must explicitly enable a role using the `SET ROLE` statement. When users enable a role in their session, they gain all of the privileges assigned to that role. Enabling a role does not affect any other roles that the users have active in their sessions. They can have multiple roles enabled simultaneously, gaining the combined privileges of all the roles they have enabled, plus any of the privileges that have been granted to them directly.

```
=> SELECT * FROM applog;  
ERROR: permission denied for relation applog  
  
=> SET ROLE logreader;
```

```
SET
```

```
=> SELECT * FROM applog;
```

id	sourceID	data	event
1	Loader	2011-03-31 11:00:38.494226	Error: Failed to open source file
2	Reporter	2011-03-31 11:00:38.494226	Warning: Low disk space on volume /scratch-a

(2 rows)

You can enable all of the roles available to your user account using the **SET ROLE ALL** statement.

```
=> SET ROLE ALL;SET  
=> SHOW ENABLED_ROLES;
```

name	setting
enabled roles	logreader, logwriter

(1 row)

See Also

- [Viewing a User's Role](#)

Disabling Roles

To disable all roles, use the **SET ROLE NONE** statement:

```
=> SET ROLE NONE;  
=> SHOW ENABLED_ROLES;  
name | setting  
-----+-----  
enabled roles |  
(1 row)
```

Viewing Enabled and Available Roles

You can list the roles you have enabled in your session using the **SHOW ENABLED ROLES** statement:

```
=> SHOW ENABLED_ROLES;  
name | setting  
-----+-----
```

```
enabled roles | logreader  
(1 row)
```

You can find the roles available to your account using the `SHOW AVAILABLE ROLES` statement:

```
Bob=> SHOW AVAILABLE_ROLES;  
      name      |      setting  
-----+-----  
available roles | logreader, logwriter  
(1 row)
```

Viewing Named Roles

To view the names of all roles users can access, along with any roles that have been assigned to those roles, query the [V_CATALOG.ROLES](#) system table.

```
=> SELECT * FROM roles;  
      role_id      |      name      |      assigned_roles  
-----+-----+-----  
45035996273704964 | public        |  
45035996273704966 | dbduser       |  
45035996273704968 | dbadmin       | dbduser*  
45035996273704972 | pseudosuperuser | dbadmin*  
45035996273704974 | logreader     |  
45035996273704976 | logwriter     |  
45035996273704978 | logadmin      | logreader, logwriter  
(7 rows)
```

Note: An asterisk (*) in the output means that role was granted WITH ADMIN OPTION.

Viewing a User's Role

The [HAS_ROLE\(\)](#) function lets you see if a role has been granted to a user.

Non-superusers can check their own role membership using `HAS_ROLE('role_name')`, but only a superuser can look up other users' memberships using the `user_name` parameter. Omitting the `user_name` parameter will return role results for the superuser who is calling the function.

How to View a User's Role

In this example, user Bob wants to see if he's been assigned the logwriter command. The output returns Boolean value `t` for true, denoting that Bob is assigned the specified logwriter role:

```
Bob=> SELECT HAS_ROLE('logwriter');
HAS_ROLE
-----
t
(1 row)
```

In this example, a superuser wants to verify that the logadmin role has been granted to user Ted:

```
dbadmin=> SELECT HAS_ROLE('Ted', 'logadmin');
```

The output returns boolean value *t* for true, denoting that Ted is assigned the specified logadmin role:

```
HAS_ROLE
-----
t
(1 row)
```

Note that if a superuser omits the `user_name` argument, the function looks up that superuser's role. The following output indicates that this superuser is not assigned the logadmin role:

```
dbadmin=> SELECT HAS_ROLE('logadmin');
HAS_ROLE
-----
f
(1 row)
```

Output of the function call with user Alice indicates that she is not granted the logadmin role:

```
dbadmin=> SELECT HAS_ROLE('Alice', 'logadmin');
HAS_ROLE
-----
f
(1 row)
```

To view additional information about users, roles and grants, you can also query the following system tables in the `V_CATALOG` schema to show directly-assigned roles:

- [ROLES](#)
- [GRANTS](#)
- [USERS](#)

Note that the system tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly). You need to call the `HAS_ROLE()` function for that information.

Users

This command returns all columns from the USERS system table:

```
=> SELECT * FROM users;
-[ RECORD 1 ]
-----+-----
user_id      | 45035996273704962
user_name    | dbadmin
is_super_user | t
profile_name | default
is_locked    | f
lock_time    |
resource_pool | general
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
run_time_cap | unlimited
all_roles    | dbadmin*, pseudosuperuser*default_roles
              | dbadmin*, pseudosuperuser*
```

Note: An asterisk (*) in table output for all_roles and default_roles columns indicates a role granted WITH ADMIN OPTION.

Roles

The following command returns all columns from the ROLES system table:

```
=> SELECT * FROM roles;
  role id      | name      | assigned_roles
-----+-----+-----
45035996273704964 | public   |
45035996273704964 | dbduser  |
45035996273704964 | dbadmin  | dbduser*
45035996273704964 | pseudosuperuser | dbadmin*
```

Grants

The following command returns all columns from the GRANTS system table:

```
=> SELECT * FROM grants;
 grantor | privileges_description | object_schema | object_name | grantee
-----+-----+-----+-----+-----
dbadmin | USAGE                  |                | public      | public
dbadmin | USAGE                  |                | v_internal  | public
dbadmin | USAGE                  |                | v_catalog   | public
dbadmin | USAGE                  |                | v_monitor   | public
(4 rows)
```

Viewing User Roles on Management Console

You can see an MC user's roles and database resources through the **MC Settings > User management** page on the Management Console interface. For more information, see [About MC Privileges and Roles](#).

Using the Administration Tools

The Vertica Administration tools allow you to easily perform administrative tasks. You can perform most Vertica database administration tasks with Administration Tools.

Run Administration Tools using the Database Administrator account on the Administration host, if possible. Make sure that no other Administration Tools processes are running.

If the Administration host is unresponsive, run Administration Tools on a different node in the cluster. That node permanently takes over the role of Administration host.

Any user can view the man page available for admintools. Enter the following:

```
man admintools
```

Running Administration Tools

As dbadmin user, you can run administration tools. The syntax follows:

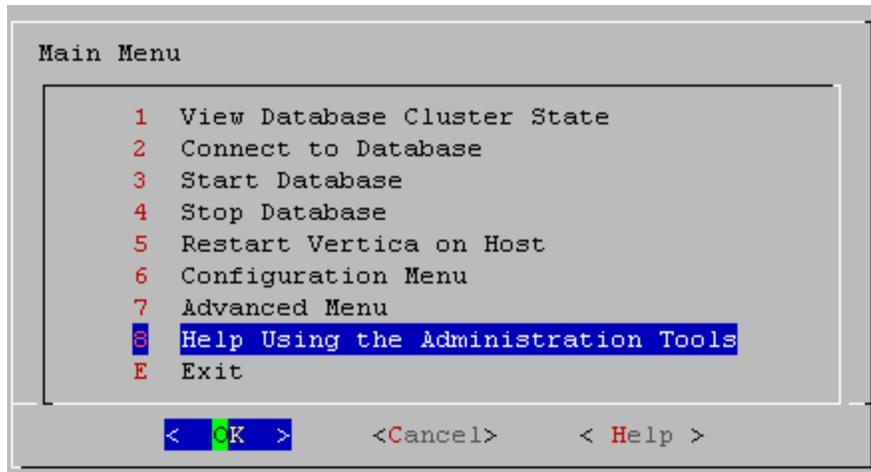
```
/opt/vertica/bin/admintools [--debug ][  
  { -h | --help }  
  | { -a | --help_all }  
  | { -t | --tool } name_of_tool [ options ]  
]
```

Options

<code>--debug</code>	If you include the debug option, Vertica logs debug information. Note: You can specify the debug option with or without naming a specific tool. If you specify debug with a specific tool, Vertica logs debug information during tool execution. If you do not specify a tool, Vertica logs debug information when you run tools through the admintools user interface.
<code>-h</code> <code>--help</code>	Outputs abbreviated help.
<code>-a</code> <code>--help_all</code>	Outputs verbose help, which lists all command-line sub-commands and options.
<code>{ -t --tool } <i>name_of_tool</i></code>	Specifies the tool to run, where <i>name_of_tool</i> is one of the tools described in the help output, and <i>options</i> are one or more

[options]	comma-delimited tool arguments. Note: Enter <code>admintools -h</code> to see the list of tools available. Enter <code>admintools -t name_of_tool --help</code> to review a specific tool's options.
-----------	--

An unqualified `admintools` command displays the Main Menu dialog box.



If you are unfamiliar with this type of interface, read [Using the Administration Tools Interface](#)

First Login as Database Administrator

The first time you log in as the Database Administrator and run the Administration Tools, the user interface displays.

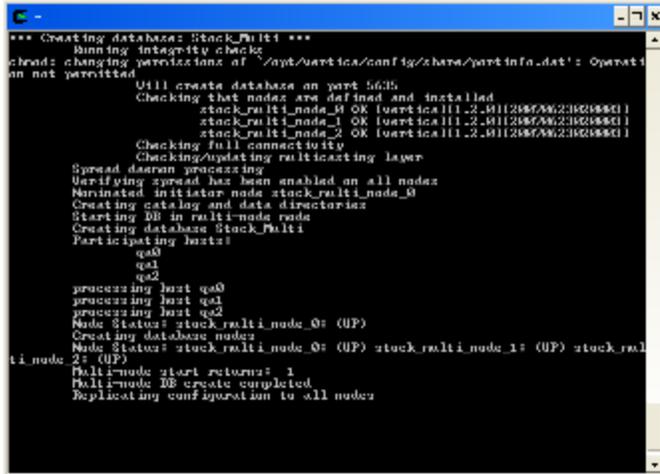
1. In the end-user license agreement (EULA) window, type `accept` to proceed.

A window displays, requesting the location of the license key file you downloaded from the Micro Focus Web site. The default path is `/tmp/vlicense.dat`.

2. Type the absolute path to your license key (for example, `/tmp/vlicense.dat`) and click **OK**.

Between Dialogs

While the Administration Tools are working, you see the command line processing in a window similar to the one shown below. Do not interrupt the processing.



```
*** Creating database: Stock_Multi ***
Running integrity check
Thread: changing permissions of /opt/vertica/config/share/partinfo.dat: Operation not permitted.
Will create database on port 5438.
Checking that nodes are defined and installed
stock_multi_node_0 OK [vertica11.2.011288736238288801]
stock_multi_node_1 OK [vertica11.2.011288736238288801]
stock_multi_node_2 OK [vertica11.2.011288736238288801]
Checking full connectivity
Checking/updating replicating layer
Spread daemon processing
Verifying spread has been enabled on all nodes
Nominated initiator node stock_multi_node_0
Creating catalog and data directories
Starting DB in multi-node mode
Creating database Stock_Multi
Participating hosts:
  qe0
  qe1
  qe2
processing host qe0
processing host qe1
processing host qe2
Node Status: stock_multi_node_0: (UP)
Creating database nodes
Node Status: stock_multi_node_0: (UP) stock_multi_node_1: (UP) stock_multi_node_2: (UP)
Multi-node start returns: 1
Multi-node DB create completed
Replicating configuration to all nodes
```

Using the Administration Tools Interface

The Vertica Administration Tools are implemented using Dialog, a graphical user interface that works in terminal (character-cell) windows. The interface responds to mouse clicks in some terminal windows, particularly local Linux windows, but you might find that it responds only to keystrokes. Thus, this section describes how to use the Administration Tools using only keystrokes.

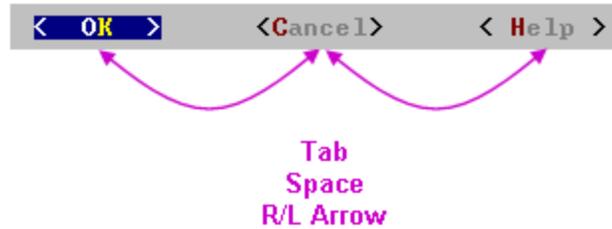
Note: This section does not describe every possible combination of keystrokes you can use to accomplish a particular task. Feel free to experiment and to use whatever keystrokes you prefer.

Enter [Return]

In all dialogs, when you are ready to run a command, select a file, or cancel the dialog, press the **Enter** key. The command descriptions in this section do not explicitly instruct you to press Enter.

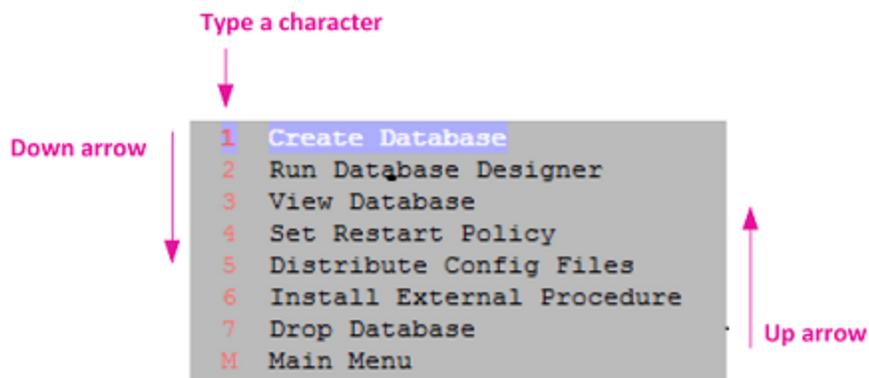
OK - Cancel - Help

The OK, Cancel, and Help buttons are present on virtually all dialogs. Use the tab, space bar, or right and left arrow keys to select an option and then press Enter. The same keystrokes apply to dialogs that present a choice of Yes or No.



Menu Dialogs

Some dialogs require that you choose one command from a menu. Type the alphanumeric character shown or use the up and down arrow keys to select a command and then press Enter.



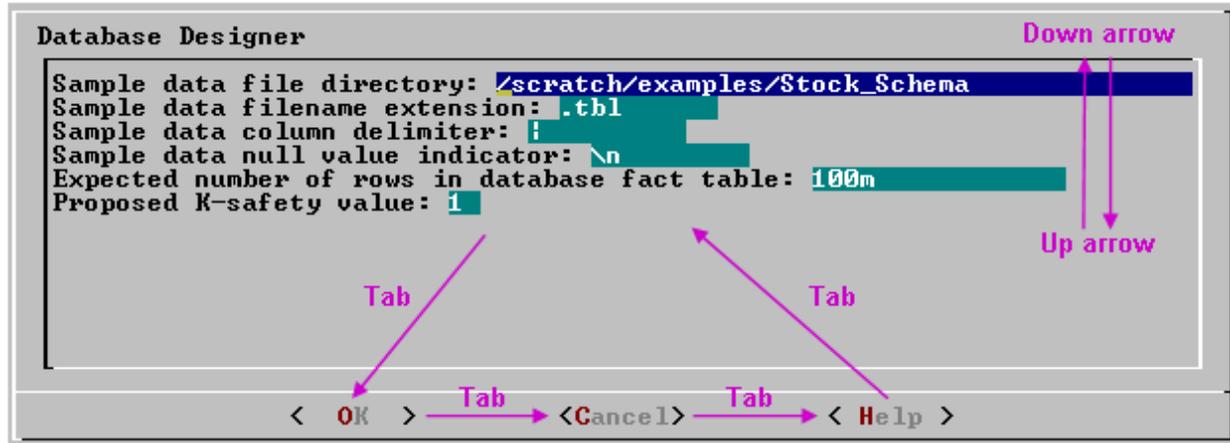
List Dialogs

In a list dialog, use the up and down arrow keys to highlight items, then use the space bar to select the items (which marks them with an X). Some list dialogs allow you to select multiple items. When you have finished selecting items, press Enter.



Form Dialogs

In a form dialog (also referred to as a dialog box), use the tab key to cycle between **OK**, **Cancel**, **Help**, and the form field area. Once the cursor is in the form field area, use the up and down arrow keys to select an individual field (highlighted) and enter information. When you have finished entering information in all fields, press Enter.



Help Buttons

Online help is provided in the form of text dialogs. If you have trouble viewing the help, see [Notes for Remote Terminal Users](#).

K-Safety Support in Administration Tools

The Administration Tools allow certain operations on a K-Safe database, even if some nodes are unresponsive.

The database must have been marked as K-Safe using the [MARK_DESIGN_KSAFE](#) function.

The following management functions within the Administration Tools are operational when some nodes are unresponsive.

Note: Vertica users can perform much of the below functionality using the Management Console interface. See [Management Console and Administration Tools](#) for details.

- View database cluster state
- Connect to database
- Start database (including manual recovery)
- Stop database
- Replace node (assuming node that is down is the one being replaced)
- View database parameters
- Upgrade license key

The following operations work with unresponsive nodes; however, you might have to repeat the operation on the failed nodes after they are back in operation:

- Distribute config files
- Install external procedure
- (Setting) database parameters

The following management functions within the Administration Tools require that all nodes be UP in order to be operational:

- Create database
- Run the Database Designer
- Drop database
- Set restart policy
- Roll back database to Last Good Epoch

Notes for Remote Terminal Users

The appearance of the graphical interface depends on the color and font settings used by your terminal window. The screen captures in this document were made using the default color and font settings in a PuTTY terminal application running on a Windows platform.

Note: If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, make sure your window is at least 81 characters wide and 23 characters high.

If you are using PuTTY, you can make the Administration Tools look like the screen captures in this document:

1. In a PuTTY window, right click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Appearance.
4. In the Font settings, click the Change... button.
5. Select Font: Courier New: Regular Size: 10
6. Click Apply.

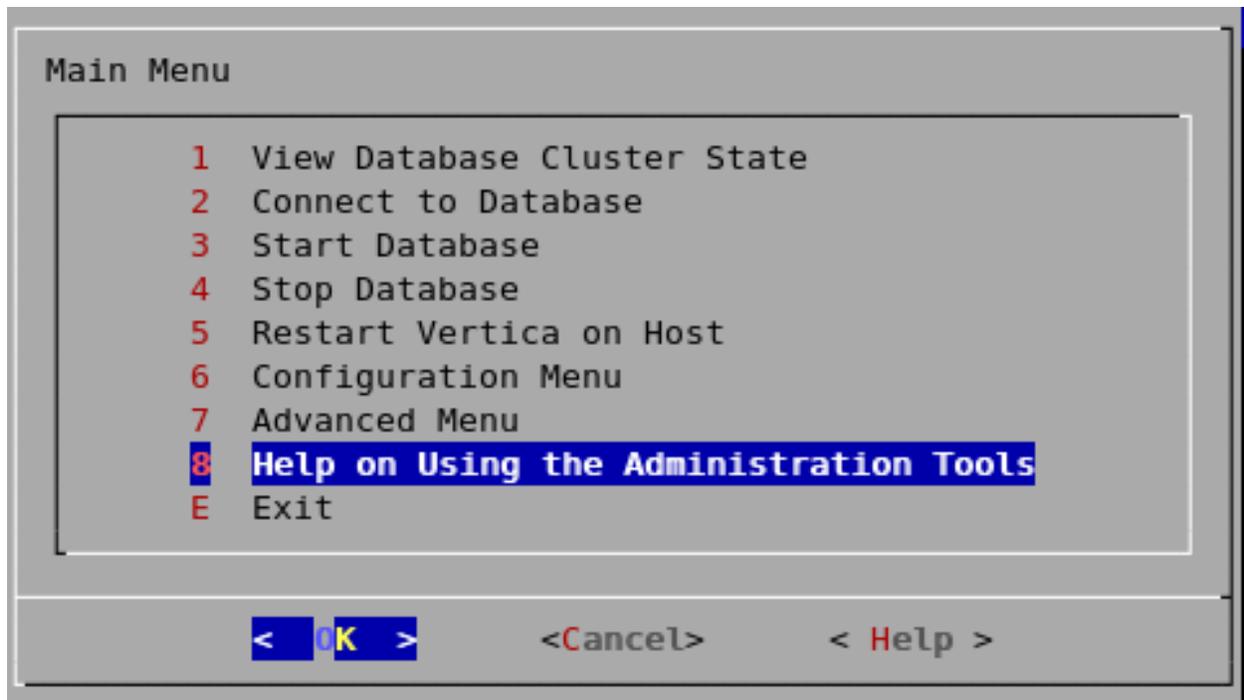
Repeat these steps for each existing session that you use to run the Administration Tools.

You can also change the translation to support UTF-8:

1. In a PuTTY window, right click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Translation.
4. In the "Received data assumed to be in which character set" drop-down menu, select UTF-8.
5. Click Apply.

Using Administration Tools Help

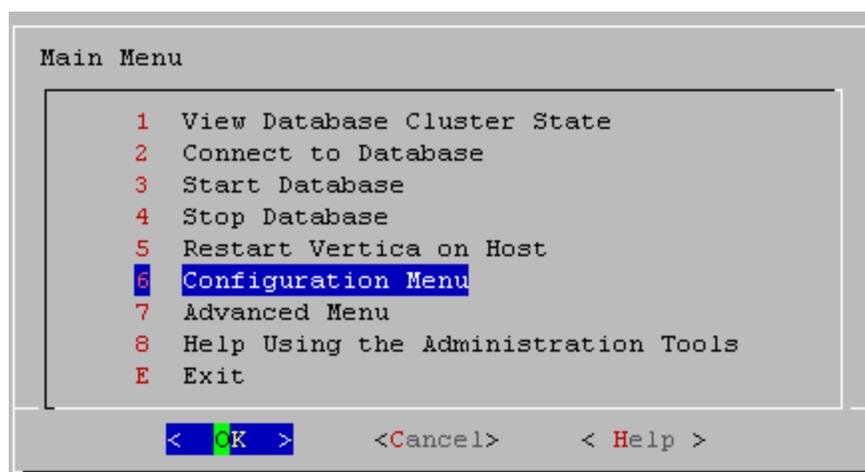
The **Help on Using the Administration Tools** command displays a help screen about using the Administration Tools.



Most of the online help in the Administration Tools is context-sensitive. For example, if you use up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

In a Menu Dialog

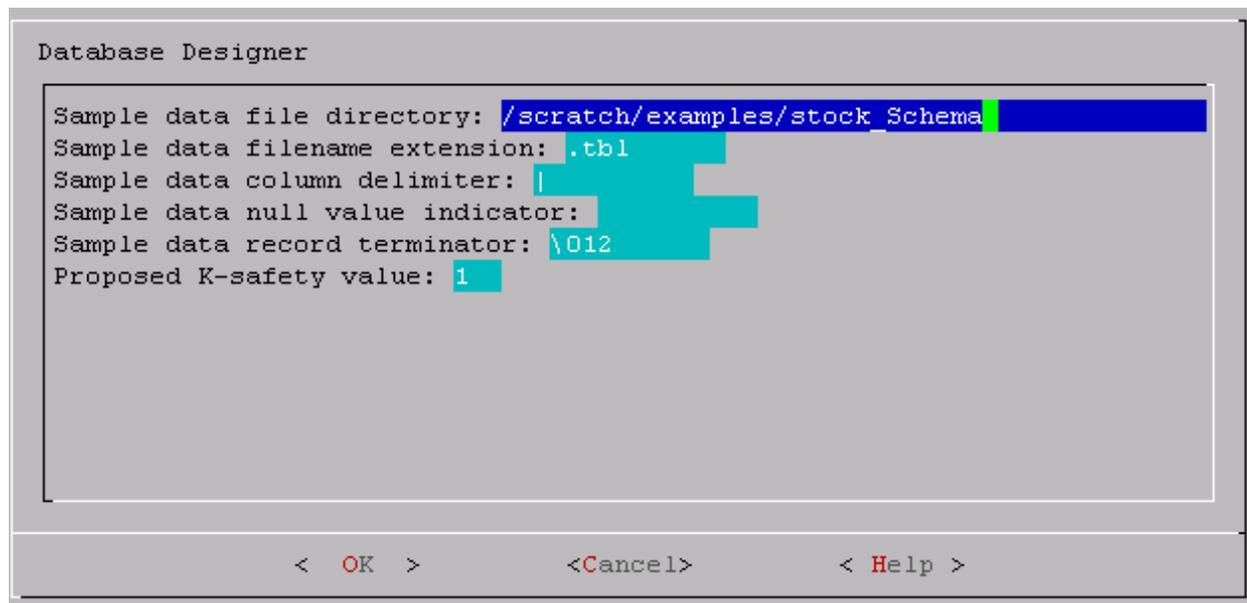
1. Use the up and down arrow keys to choose the command for which you want help.



2. Use the Tab key to move the cursor to the Help button.
3. Press Enter (Return).

In a Dialog Box

1. Use the up and down arrow keys to choose the field on which you want help.



2. Use the Tab key to move the cursor to the Help button.
3. Press Enter (Return).

Scrolling

Some help files are too long for a single screen. Use the up and down arrow keys to scroll through the text.

Password Authentication

When you create a new user with the [CREATE USER](#) command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the [ALTER USER](#) command.

See [Security and Authentication](#) for more information about controlling database authorization through passwords.

Tip: Unless the database is used solely for evaluation purposes, Micro Focus recommends that all database users have encrypted passwords.

Distributing Changes Made to the Administration Tools Metadata

Administration Tools-specific metadata for a failed node will fall out of synchronization with other cluster nodes if you make the following changes:

- Modify the restart policy
- Add one or more nodes
- Drop one or more nodes.

When you restore the node to the database cluster, you can use the Administration Tools to update the node with the latest Administration Tools metadata:

1. Log on to a host that contains the metadata you want to transfer and start the Administration Tools. (See [Using the Administration Tools](#).)
2. On the **Main Menu** in the Administration Tools, select **Configuration Menu** and click **OK**.
3. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
4. Select **AdminTools Meta-Data**.

The Administration Tools metadata is distributed to every host in the cluster.

5. [Restart the database](#).

Administration Tools and Management Console

You can perform most database administration tasks using the Administration Tools, but you have the additional option of using the more visual and dynamic Management Console.

The following table compares the functionality available in both interfaces. Continue to use Administration Tools and the command line to perform actions not yet supported by Management Console.

Vertica Functionality	Management Console	Administration Tools
Use a Web interface for the administration of Vertica	Yes	No
Manage/monitor one or more databases and clusters through a UI	Yes	No
Manage multiple databases on different clusters	Yes	Yes
View database cluster state	Yes	Yes
View multiple cluster states	Yes	No
Connect to the database	Yes	Yes
Start/stop an existing database	Yes	Yes
Stop/restart Vertica on host	Yes	Yes
Kill an Vertica process on host	No	Yes
Create one or more databases	Yes	Yes
View databases	Yes	Yes
Remove a database from view	Yes	No
Drop a database	Yes	Yes
Create a physical schema design (Database Designer)	Yes	Yes
Modify a physical schema design (Database Designer)	Yes	Yes
Set the restart policy	No	Yes
Roll back database to the Last Good Epoch	No	Yes
Manage clusters (add, replace, remove hosts)	Yes	Yes
Rebalance data across nodes in the database	Yes	Yes

Vertica Functionality	Management Console	Administration Tools
Configure database parameters dynamically	Yes	No
View database activity in relation to physical resource usage	Yes	No
View alerts and messages dynamically	Yes	No
View current database size usage statistics	Yes	No
View database size usage statistics over time	Yes	No
Upload/upgrade a license file	Yes	Yes
Warn users about license violation on login	Yes	Yes
Create, edit, manage, and delete users/user information	Yes	No
Use LDAP to authenticate users with company credentials	Yes	Yes
Manage user access to MC through roles	Yes	No
Map Management Console users to an Vertica database	Yes	No
Enable and disable user access to MC and/or the database	Yes	No
Audit user activity on database	Yes	No
Hide features unavailable to a user through roles	Yes	No
Generate new user (non-LDAP) passwords	Yes	No

Management Console Provides some, but Not All of the Functionality Provided By the Administration Tools. MC Also Provides Functionality Not Available in the Administration Tools.

See Also

- [Monitoring Vertica Using Management Console](#)

Administration Tools Reference

With Vertica Administration Tools, you can perform the following tasks:

- [View the database cluster state](#)
- [Connect to the database](#)
- [Start the database](#)
- [Stop the database](#)
- [Write scripts](#)

Viewing Database Cluster State

This tool shows the current state of the nodes in the database.

1. On the Main Menu, select **View Database Cluster State**, and click **OK**.
The normal state of a running database is ALL UP. The normal state of a stopped database is ALL DOWN.

DB	Host	State
Clickstream_Schema	ALL	DOWN
CreditHistory_Schema	ALL	DOWN
Retail_Schema	ALL	DOWN
Stock_Schema	ALL	UP
Telecom_Schema	ALL	DOWN

2. If some hosts are UP and some DOWN, restart the specific host that is down using **Restart Vertica on Host** from the Administration Tools, or you can start the database as described in [Starting and Stopping the Database](#) (unless you have a known node failure and want to continue in that state.)

DB	Host	State
Clickstream_Schema	ALL	DOWN
CreditHistory_Schema	ALL	DOWN
Retail_Schema	ALL	DOWN
Stock_Schema	host01	UP
Stock_Schema	host02	UP
Stock_Schema	host03	DOWN
Stock_Schema	host04	UP
Telecom_Schema	ALL	DOWN

Nodes shown as INITIALIZING or RECOVERING indicate that [Failure Recovery](#) is in progress.

Nodes in other states (such as `NEEDS_CATCHUP`) are transitional and can be ignored unless they persist.

See Also

- [Advanced Menu Options](#)

Connecting to the Database

This tool connects to a running database with `vsq`. You can use the Administration Tools to connect to a database from any node within the database while logged in to any user account with access privileges. You cannot use the Administration Tools to connect from a host that is not a database node. To connect from other hosts, run `vsq` as described in [Connecting from the Command Line](#).

1. On the Main Menu, click **Connect to Database**, and then click **OK**.
2. Supply the database password if asked:

```
Password:
```

When you create a new user with the [CREATE USER](#) command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the [ALTER USER](#) command.

The Administration Tools connect to the database and transfer control to `vsq`.

```
Welcome to vsq, the Vertica Analytic Database interactive terminal.  
Type: \h or \? for help with vsq commands  
      \g or terminate with semicolon to execute query  
      \q to quit  
  
=>
```

See [Using vsq](#) for more information.

Note: After entering your password, you may be prompted to change your password if it has expired. See [Implementing Client Authentication](#) for details of password security.

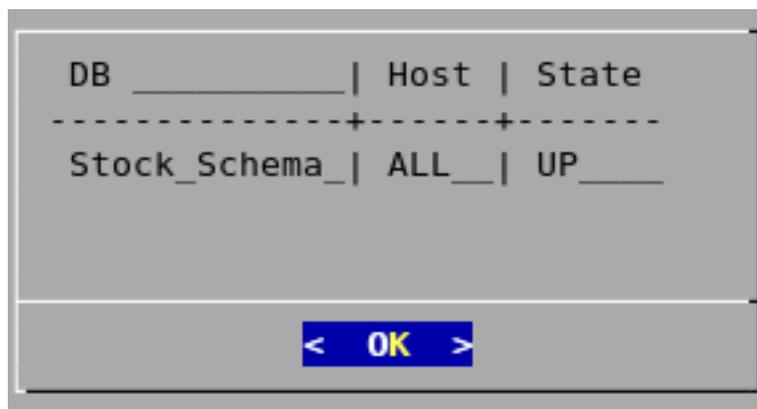
See Also

- [CREATE USER](#)
- [ALTER USER](#)

Restarting Vertica on Host

This tool restarts the Vertica process on one or more nodes in a running database. Use this tool when a cluster host reboots while the database is running. The spread daemon starts automatically but the Vertica process does not, so the node does not automatically rejoin the cluster.

1. On the Main Menu, select **View Database Cluster State**, and click **OK**.
2. If one or more nodes are down, select **Restart Vertica on Host**, and click **OK**.
3. Select the database that contains the host that you want to restart, and click **OK**.
4. Select the Host to restart, and click **OK**.
5. Select **View Database Cluster State** again to verify all nodes are up.



Configuration Menu Item

The Configuration Menu includes:

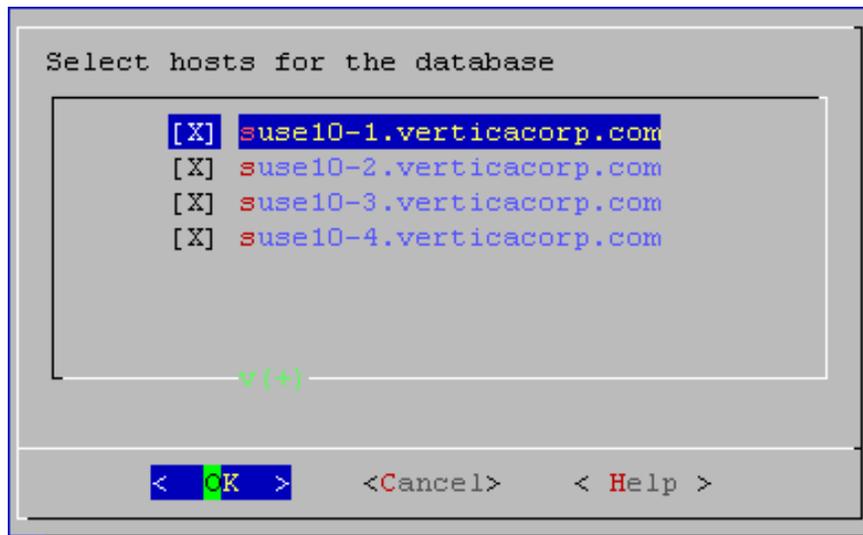
Creating a Database

1. On the **Configuration Menu**, click **Create Database** and then click **OK**.
2. Enter the name of the database and an optional comment. Click **OK**.
3. Enter a password. See [Creating a Database Name and Password](#) for rules.

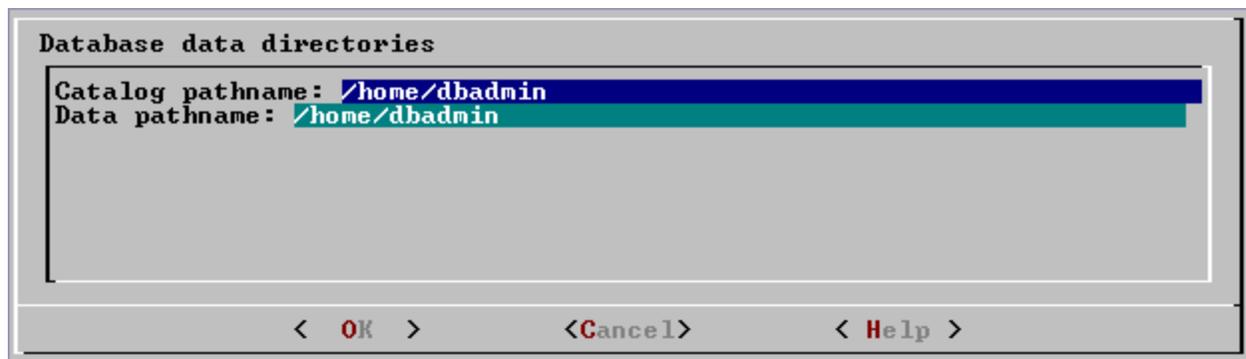
If you do not enter a password, you are prompted to indicate whether you want to enter a password. Click **Yes** to enter a password or **No** to create a database without a superuser password.

Caution: If you do not enter a password at this point, superuser password is set to empty. Unless the database is for evaluation or academic purposes, Micro Focus strongly recommends that you enter a superuser password.

4. If you entered a password, enter the password again.
5. Select the hosts to include in the database. The hosts in this list are the ones that were specified at installation time (`install_vertica -s`).



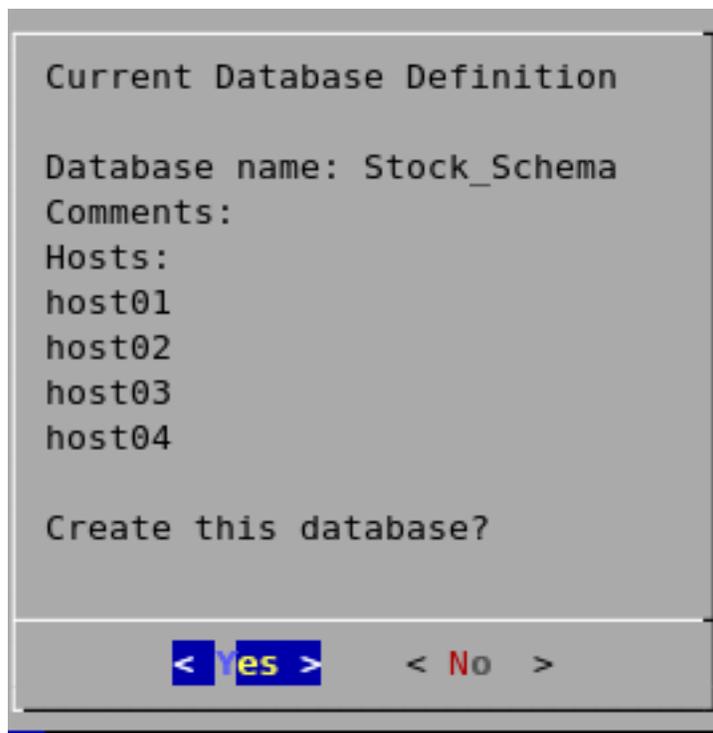
6. Specify the directories in which to store the catalog and data files.



Note: Catalog and data paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.

Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

7. Check the current database definition for correctness, and click Yes to proceed.



8. A message indicates that you have successfully created a database. Click **OK**.

Dropping a Database

This tool drops an existing database. Only the Database Administrator is allowed to drop a database.

1. [Stop the database](#).
2. On the **Configuration Menu**, click **Drop Database** and then click **OK**.
3. Select the database to drop and click **OK**.
4. Click **Yes** to confirm that you want to drop the database.
5. Type **yes** and click **OK** to reconfirm that you really want to drop the database.
6. A message indicates that you have successfully dropped the database. Click **OK**.

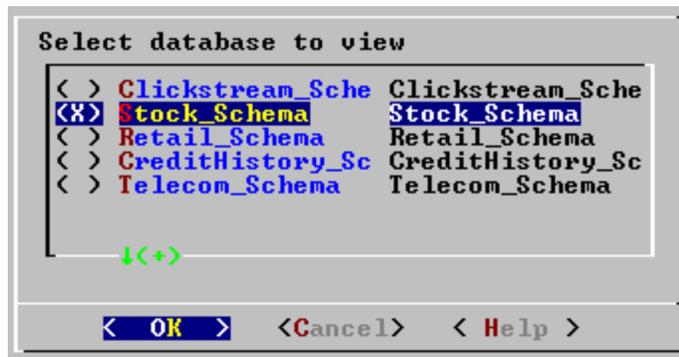
When Vertica drops the database, it also automatically drops the node definitions that refer to the database . The following exceptions apply:

- Another database uses a node definition. If another database refers to any of these node definitions, none of the node definitions are dropped.
- A node definition is the only node defined for the host. (Vertica uses node definitions to locate hosts that are available for database creation, so removing the only node defined for a host would make the host unavailable for new databases.)

Viewing a Database

This tool displays the characteristics of an existing database.

1. On the **Configuration Menu**, select **View Database** and click **OK**.
2. Select the database to view.



3. Vertica displays the following information about the database:

- The name of the database.
- The name and location of the log file for the database.
- The hosts within the database cluster.
- The value of the restart policy setting.

Note: This setting determines whether nodes within a K-Safe database are restarted when they are rebooted. See [Setting the Restart Policy](#).

- The database port.
- The name and location of the catalog directory.

Setting the Restart Policy

The Restart Policy enables you to determine whether or not nodes in a K-Safe database are automatically restarted when they are rebooted. Since this feature does not automatically restart nodes if the entire database is DOWN, it is not useful for databases that are not K-Safe.

To set the Restart Policy for a database:

1. Open the Administration Tools.
2. On the Main Menu, select **Configuration Menu**, and click **OK**.
3. In the Configuration Menu, select **Set Restart Policy**, and click **OK**.
4. Select the database for which you want to set the Restart Policy, and click **OK**.
5. Select one of the following policies for the database:
 - **Never** — Nodes are never restarted automatically.
 - **K-Safe** — Nodes are automatically restarted if the database cluster is still UP. This is the default setting.
 - **Always** — Node on a single node database is restarted automatically.

Note: Always does not work if a single node database was not shutdown cleanly or crashed.

6. Click **OK**.

Best Practice for Restoring Failed Hardware

Following this procedure will prevent Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

1. Reboot the machine into runlevel 1, which is a root and console-only mode.

Runlevel 1 prevents network connectivity and keeps Vertica from attempting to reconnect to the cluster.

2. In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

3. Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

At this point, the network activates, and Vertica rejoins the cluster and automatically recovers any missing data. Note that, on a single-node database, if any files that were associated with a projection have been deleted or corrupted, Vertica will delete all files associated with that projection, which could result in data loss.

Installing External Procedure Executable Files

1. Run the Administration Tools.

```
$ /opt/vertica/bin/adminTools
```

2. On the AdminTools **Main Menu**, click **Configuration Menu**, and then click **OK**.
3. On the **Configuration Menu**, click **Install External Procedure** and then click **OK**.
4. Select the database on which you want to install the external procedure.
5. Either select the file to install or manually type the complete file path, and then click **OK**.
6. If you are not the superuser, you are prompted to enter your password and click **OK**.

The Administration Tools automatically create the `<database_catalog_path>/procedures` directory on each node in the database and installs the external procedure in these directories for you.

7. Click **OK** in the dialog that indicates that the installation was successful.

Advanced Menu Options

This Advanced Menu includes:

Rolling Back the Database to the Last Good Epoch

Vertica provides the ability to roll the entire database back to a specific epoch primarily to assist in the correction of human errors during data loads or other accidental corruptions. For example, suppose that you have been performing a bulk load and the cluster went down during a particular [COPY](#) command. You might want to discard all epochs back to the point at which the previous COPY command committed and run the one that did not finish again. You can determine that point by examining the log files (see [Monitoring the Log Files](#)).

1. On the Advanced Menu, select **Roll Back Database to Last Good Epoch**.
2. Select the database to roll back. The database must be stopped.
3. Accept the suggested restart epoch or specify a different one.
4. Confirm that you want to discard the changes after the specified epoch.

The database restarts successfully.

Important: The default value of `HistoryRetentionTime` is 0, which means that Vertica only keeps historical data when nodes are down. This settings prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch. Vertica cannot roll back to an epoch that precedes the AHM.

If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data. For example:

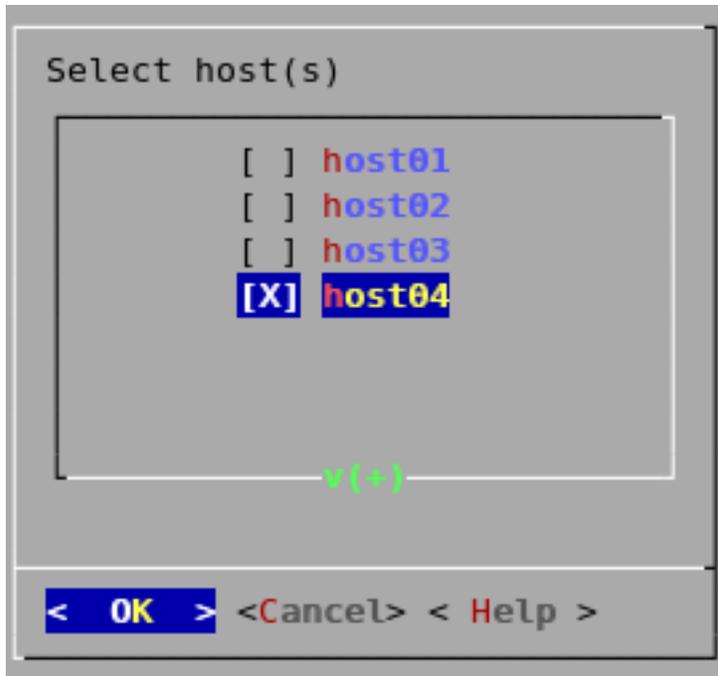
```
=> ALTER DATABASE mydb SET HistoryRetentionTime = 86400;
```

Stopping Vertica on Host

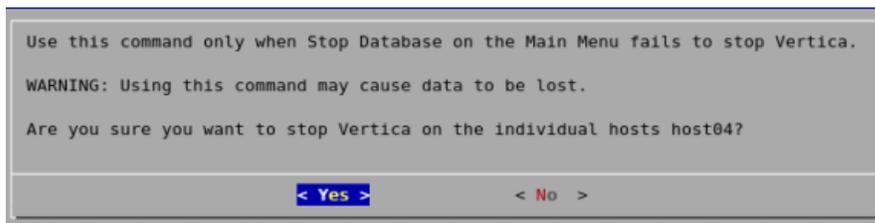
This command attempts to gracefully shut down the Vertica process on a single node.

Caution: Do not use this command to shut down the entire cluster. Instead, [stop the database](#) to perform a clean shutdown that minimizes data loss.

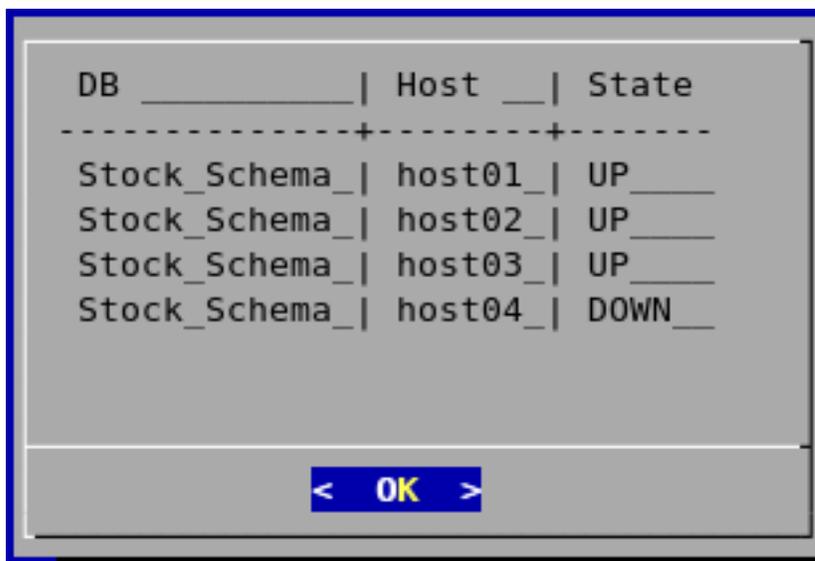
1. On the Advanced Menu, select **Stop Vertica on Host** and click **OK**.
2. Select the hosts to stop.



3. Confirm that you want to stop the hosts.



If the command succeeds [View Database Cluster State](#) shows that the selected hosts are DOWN.



DB _____	Host ____	State
Stock_Schema_	host01_	UP_____
Stock_Schema_	host02_	UP_____
Stock_Schema_	host03_	UP_____
Stock_Schema_	host04_	DOWN_____

< OK >

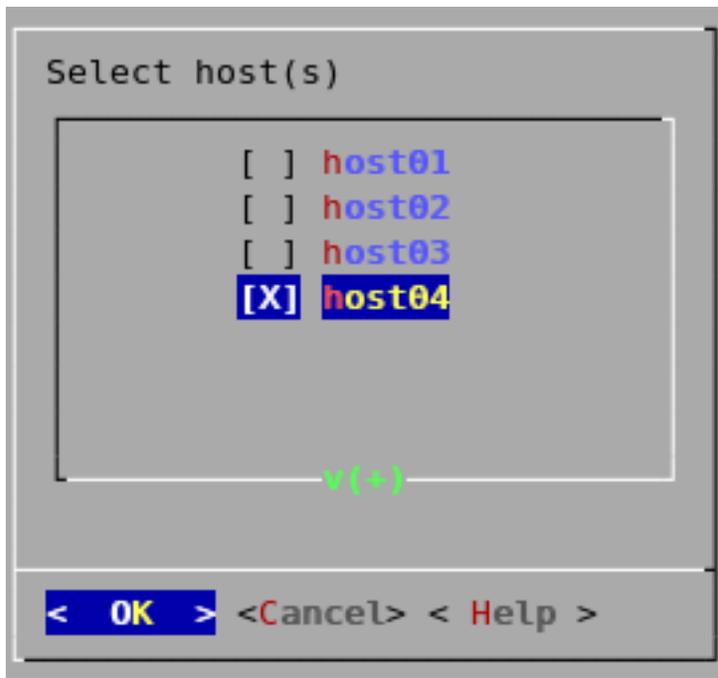
If the command fails to stop any selected nodes, proceed to [Killing Vertica Process on Host](#).

Killing the Vertica Process on Host

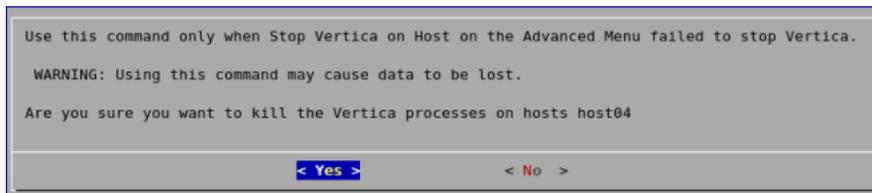
This command sends a kill signal to the Vertica process on a node.

Caution: Use this command only after you tried to [stop the database](#) and [stop Vertica on a node](#) and both were unsuccessful.

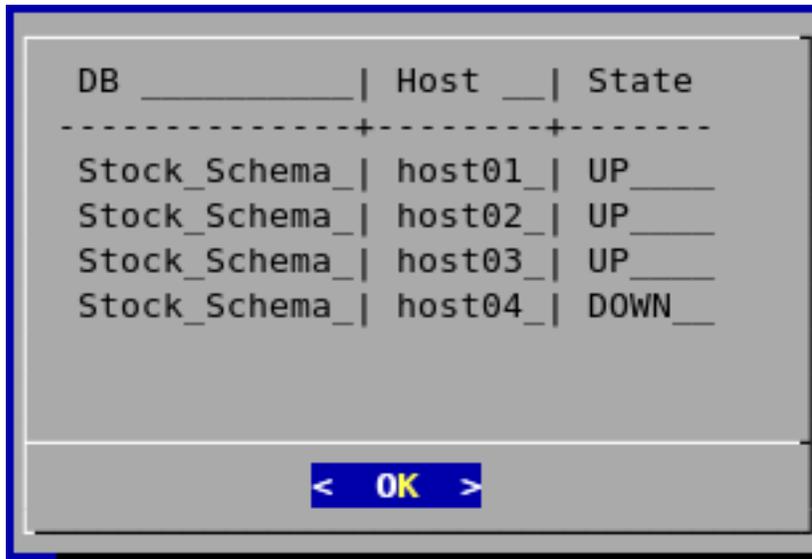
1. On the Advanced menu, select **Kill Vertica Process on Host** and click **OK**.
2. Select the hosts on which to kills the Vertica process.



3. Confirm that you want to stop the processes.



4. If the command succeeds, [View Database Cluster State](#) shows that the selected hosts are DOWN.



DB _____	Host ___	State
Stock_Schema_	host01_	UP_____
Stock_Schema_	host02_	UP_____
Stock_Schema_	host03_	UP_____
Stock_Schema_	host04_	DOWN____

< OK >

Upgrading a Vertica License Key

The following steps are for licensed Vertica users. Completing the steps copies a license key file into the database. See [Managing Licenses](#) for more information.

1. On the Advanced menu select **Upgrade License Key** . Click **OK**.
2. Select the database for which to upgrade the license key.
3. Enter the absolute pathname of your downloaded license key file (for example, /tmp/vlicense.dat). Click **OK**.
4. Click OK when you see a message indicating that the upgrade succeeded.

Note: If you are using Vertica Community Edition, follow the instructions in [Vertica License Renewals or Upgrades](#) for instructions to upgrade to a Vertica Premium Edition license key.

Managing Clusters

Cluster Management lets you add, replace, or remove hosts from a database cluster. These processes are usually part of a larger process of [adding](#), [removing](#), or [replacing](#) a database node.

Note: View the database state to verify that it is running. See [View Database Cluster State](#). If the database isn't running, restart it. See [Start the Database](#).

Using Cluster Management

To use Cluster Management:

1. From the **Main Menu**, select **Advanced Menu**, and then click **OK**.
2. In the **Advanced Menu**, select **Cluster Management**, and then click **OK**.
3. Select one of the following, and then click **OK**.
 - **Add Hosts to Database:** See [Adding Hosts to a Database](#).
 - **Re-balance Data:** See [Rebalancing Data](#).
 - **Replace Host:** See [Replacing Hosts](#).
 - **Remove Host from Database:** See [Removing Hosts from a Database](#).

Using Administration Tools

The **Help Using the Administration Tools** command displays a help screen about using the Administration Tools.

Most of the online help in the Administration Tools is context-sensitive. For example, if you use the up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

Administration Tools Metadata

The Administration Tools configuration data (metadata) contains information that databases need to start, such as the hostname/IP address of each participating host in the database cluster.

To facilitate hostname resolution within the Administration Tools, at the command line, and inside the installation utility, Vertica enforces all hostnames you provide through the Administration Tools to use IP addresses:

- **During installation**

Vertica immediately converts any hostname you provide through command line options `--hosts`, `--add-hosts` or `--remove-hosts` to its IP address equivalent.

- If you provide a hostname during installation that resolves to multiple IP addresses (such as in multi-homed systems), the installer prompts you to choose one IP address.
- Vertica retains the name you give for messages and prompts only; internally it stores these hostnames as IP addresses.

- **Within the Administration Tools**

All hosts are in IP form to allow for direct comparisons (for example db = database = database.example.com).

- **At the command line**

Vertica converts any hostname value to an IP address that it uses to look up the host in the configuration metadata. If a host has multiple IP addresses that are resolved, Vertica tests each IP address to see if it resides in the metadata, choosing the first match. No match indicates that the host is not part of the database cluster.

Metadata is more portable because Vertica does not require the names of the hosts in the cluster to be exactly the same when you install or upgrade your database.

Writing Administration Tools Scripts

You can invoke most Administration Tools from the command line or a shell script.

Syntax

```
/opt/vertica/bin/admintools {  
  { -h | --help }  
  | { -a | --help_all }  
  | { [--debug] { -t | --tool } toolname [ tool-args ] }  
}
```

Note: For convenience, add `/opt/vertica/bin` to your search path.

Parameters

-h -help	Outputs abbreviated help.
-a	Outputs verbose help, which lists all command-line sub-commands

<code>-help_all</code>	and options.
<code>[debug] { -t -tool } toolname [args]</code>	Specifies the tool to run, where <i>toolname</i> is one of the tools listed in the help output described below, and <i>args</i> is one or more comma-delimited <i>toolname</i> arguments. If you include the debug option, Vertica logs debug information during tool execution.

Tools

To return a list of all available tools, enter `admintools -h` at a command prompt.

Note: To create a database or password, see [Creating a Database Name and Password](#) for naming rules.

To display help for a specific tool and its options or commands, qualify the specified tool name with `--help` or `-h`, as shown in the example below:

```
$ admintools -t connect_db --help
Usage: connect_db [options]

Options:
  -h, --help            show this help message and exit
  -d DB, --database=DB  Name of database to connect
  -p DBPASSWORD, --password=DBPASSWORD
                        Database password in single quotes
```

To list all available tools and their commands and options in individual help text, enter `admintools -a`.

```
$ admintools -a
Usage:
  adminTools [-t | --tool] toolName [options]
Valid tools are:
  command_host
  connect_db
  create_db
  database_parameters
  db_add_node
  db_remove_node
  db_replace_node
  db_status
  distribute_config_files
  drop_db
  host_to_node
  install_package
  install_procedure
  kill_host
  kill_node
```

```
license_audit  
list_allnodes  
list_db  
list_host  
list_node  
list_packages  
logrotate  
node_map  
rebalance_data  
restart_db  
restart_node  
return_epoch  
set_restart_policy  
set_ssl_params  
show_active_db  
start_db  
stop_db  
stop_host  
stop_node  
uninstall_package  
upgrade_license_key  
view_cluster
```

Usage: command_host [options]

Options:

```
-h, --help          show this help message and exit  
-c CMD, --command=CMD  
                    Command to run
```

Usage: connect_db [options]

Options:

```
-h, --help          show this help message and exit  
-d DB, --database=DB Name of database to connect  
-p DBPASSWORD, --password=DBPASSWORD  
                    Database password in single quotes
```

Usage: create_db [options]

Options:

```
-h, --help          show this help message and exit  
-D DATA, --data_path=DATA  
                    Path of data directory[optional] if not using compat21  
-c CATALOG, --catalog_path=CATALOG  
                    Path of catalog directory[optional] if not using  
                    compat21  
--compat21         (deprecated) Use Vertica 2.1 method using node names  
                    instead of hostnames  
-d DB, --database=DB Name of database to be created  
-l LICENSEFILE, --license=LICENSEFILE  
                    Database license [optional]  
-p DBPASSWORD, --password=DBPASSWORD  
                    Database password in single quotes [optional]  
-P POLICY, --policy=POLICY  
                    Database restart policy [optional]  
-s NODES, --hosts=NODES  
                    comma-separated list of hosts to participate in  
                    database
```

```
--skip-fs-checks      Skip file system checks while creating a database (not
                      recommended).
-----
Usage: database_parameters [options]

Options:
-h, --help            show this help message and exit
-d DB, --database=DB  Name of database
-P PARAMETER, --parameter=PARAMETER
                      Database parameter
-c COMPONENT, --component=COMPONENT
                      Component[optional]
-s SUBCOMPONENT, --subcomponent=SUBCOMPONENT
                      Sub Component[optional]
-p PASSWORD, --password=PASSWORD
                      Database password[optional]
-----
Usage: db_add_node [options]

Options:
-h, --help            show this help message and exit
-d DB, --database=DB  Name of database to be restarted
-s HOSTS, --hosts=HOSTS
                      Comma separated list of hosts to add to database
-p DBPASSWORD, --password=DBPASSWORD
                      Database password in single quotes
-a AHOSTS, --add=AHOSTS
                      Comma separated list of hosts to add to database
-i, --noprompts       do not stop and wait for user input(default false)
--compat21            (deprecated) Use Vertica 2.1 method using node names
                      instead of hostnames
--skip-fs-checks     Skip file system checks while adding nodes (not
                      recommended).
-----
Usage: db_remove_node [options]

Options:
-h, --help            show this help message and exit
-d DB, --database=DB  Name of database to be modified
-s HOSTS, --hosts=HOSTS
                      Name of the host to remove from the db
-p DBPASSWORD, --password=DBPASSWORD
                      Database password in single quotes
-i, --noprompts       do not stop and wait for user input (default false)
--compat21            (deprecated) Use Vertica 2.1 method using node names
                      instead of hostnames
-----
Usage: db_replace_node [options]

Options:
-h, --help            show this help message and exit
-d DB, --database=DB  Name of database to be restarted
-o ORIGINAL, --original=ORIGINAL
                      Name of host you wish to replace
-n NEWHOST, --new=NEWHOST
                      Name of the replacement host
-p DBPASSWORD, --password=DBPASSWORD
                      Database password in single quotes
-i, --noprompts       do not stop and wait for user input(default false)
--skip-fs-checks     Skip file system checks while replacing nodes (not
```

```
recommended).
-----
Usage: db_status [options]

Options:
  -h, --help          show this help message and exit
  -s STATUS, --status=STATUS
                      Database status UP,DOWN or ALL(list running dbs -
                      UP,list down dbs - DOWN list all dbs - ALL
-----
Usage: distribute_config_files
Sends admintools.conf from local host to all other hosts in the cluster

Options:
  -h, --help          show this help message and exit
-----
Usage: drop_db [options]

Options:
  -h, --help          show this help message and exit
  -d DB, --database=DB Database to be dropped
-----
Usage: host_to_node [options]

Options:
  -h, --help          show this help message and exit
  -s HOST, --host=HOST comma separated list of hostnames which is to be
                      converted into its corresponding nodenames
  -d DB, --database=DB show only node/host mapping for this database.
-----
Usage: admintools -t install_package --package PACKAGE -d DB -p PASSWORD

Examples:
admintools -t install_package -d mydb -p 'mypasswd' --package default
# (above) install all default packages that aren't currently installed

admintools -t install_package -d mydb -p 'mypasswd' --package default --force-reinstall
# (above) upgrade (re-install) all default packages to the current version

admintools -t install_package -d mydb -p 'mypasswd' --package hcat
# (above) install package hcat

See also: admintools -t list_packages

Options:
  -h, --help          show this help message and exit
  -d DBNAME, --dbname=DBNAME
                      database name
  -p PASSWORD, --password=PASSWORD
                      database admin password
  -P PACKAGE, --package=PACKAGE
                      specify package or 'all' or 'default'
  --force-reinstall  Force a package to be re-installed even if it is
                      already installed.
-----
Usage: install_procedure [options]

Options:
  -h, --help          show this help message and exit
  -d DBNAME, --database=DBNAME
```

```

        Name of database for installed procedure
-f PROCPATH, --file=PROCPATH
        Path of procedure file to install
-p OWNERPASSWORD, --password=OWNERPASSWORD
        Password of procedure file owner
-----
Usage: kill_host [options]

Options:
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed using a SIGKILL signal
--compat21          (deprecated) Use Vertica 2.1 method using node names
                    instead of hostnames
-----
Usage: kill_node [options]

Options:
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed using a SIGKILL signal
--compat21          (deprecated) Use Vertica 2.1 method using node names
                    instead of hostnames
-----
Usage: license_audit --dbname DB_NAME [OPTIONS]
Runs audit and collects audit results.

Options:
-h, --help          show this help message and exit
-d DATABASE, --database=DATABASE
                    Name of the database to retrieve audit results
-p PASSWORD, --password=PASSWORD
                    Password for database admin
-q, --quiet          Do not print status messages.
-f FILE, --file=FILE Output results to FILE.
-----
Usage: list_allnodes [options]

Options:
-h, --help          show this help message and exit
-----
Usage: list_db [options]

Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database to be listed
-----
Usage: list_host [options]

Options:
-h, --help          show this help message and exit
-----
Usage: list_node [options]

Options:
-h, --help          show this help message and exit
-n NODENAME, --node=NODENAME
                    Name of the node to be listed
```

```
-----
Usage: admintools -t list_packages [OPTIONS]

Examples:
admintools -t list_packages                # lists all available packages
admintools -t list_packages --package all  # lists all available packages
admintools -t list_packages --package default # list all packages installed by default
admintools -t list_packages -d mydb --password 'mypasswd' # list the status of all packages in mydb

Options:
-h, --help          show this help message and exit
-d DBNAME, --dbname=DBNAME
                    database name
-p PASSWORD, --password=PASSWORD
                    database admin password
-P PACKAGE, --package=PACKAGE
                    specify package or 'all' or 'default'
-----

Usage: logrotate [options]

Options:
-h, --help          show this help message and exit
-d DBNAME, --dbname=DBNAME
                    database name
-r ROTATION, --rotation=ROTATION
                    set how often the log is rotated.[
                    daily|weekly|monthly ]
-s MAXLOGSZ, --maxsize=MAXLOGSZ
                    set maximum log size before rotation is forced.
-k KEEP, --keep=KEEP set # of old logs to keep
-----

Usage: node_map [options]

Options:
-h, --help          show this help message and exit
-d DB, --database=DB List only data for this database.
-----

Usage: rebalance_data [options]

Options:
-h, --help          show this help message and exit
-d DBNAME, --dbname=DBNAME
                    database name
-k KSAFETY, --ksafety=KSAFETY
                    specify the new k value to use
-p PASSWORD, --password=PASSWORD
--script            Don't re-balance the data, just provide a script for
                    later use.
-----

Usage: restart_db [options]

Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database to be restarted
-e EPOCH, --epoch=EPOCH
                    Epoch at which the database is to be restarted. If
                    'last' is given as argument the db is restarted from
                    the last good epoch.
-p DBPASSWORD, --password=DBPASSWORD
                    Database password in single quotes
```

```
--timeout=NONINTERACTIVE_TIMEOUT
        set a timeout (in seconds) to wait for actions to
        complete ('never') will wait forever (implicitly sets
        -i)
-i, --noprompts      do not stop and wait for user input(default false)
-----
Usage: restart_node [options]

Options:
-h, --help          show this help message and exit
-s NODES, --hosts=NODES
                    comma-separated list of hosts to be restarted
-d DB, --database=DB Name of database whose node is to be restarted
-p DBPASSWORD, --password=DBPASSWORD
                    Database password in single quotes
--timeout=NONINTERACTIVE_TIMEOUT
        set a timeout (in seconds) to wait for actions to
        complete ('never') will wait forever (implicitly sets
        -i)
-i, --noprompts    do not stop and wait for user input(default false)
-F, --force        force the node to start and auto recover if necessary
--compat21         (deprecated) Use Vertica 2.1 method using node names
                    instead of hostnames
-----
Usage: return_epoch [options]

Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database
-----
Usage: set_restart_policy [options]

Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database for which to set policy
-p POLICY, --policy=POLICY
                    Restart policy: ('never', 'ksafe', 'always')
-----
Usage: set_ssl_params [options]

Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database whose parameters will be set
-k KEYFILE, --ssl-key-file=KEYFILE
                    Path to SSL private key file
-c CERTFILE, --ssl-cert-file=CERTFILE
                    Path to SSL certificate file
-a CAFILE, --ssl-ca-file=CAFILE
                    Path to SSL CA file
-p DBPASSWORD, --password=DBPASSWORD
                    Database password in single quotes
-----
Usage: show_active_db [options]

Options:
-h, --help          show this help message and exit
-----
Usage: start_db [options]

Options:
```

```
-h, --help          show this help message and exit
-d DB, --database=DB  Name of database to be started
-p DBPASSWORD, --password=DBPASSWORD
                    Database password in single quotes
--timeout=NONINTERACTIVE_TIMEOUT
                    set a timeout (in seconds) to wait for actions to
                    complete ('never') will wait forever (implicitly sets
                    -i)
-i, --noprompts     do not stop and wait for user input(default false)
-F, --force         force the database to start at an epoch before data
                    consistency problems were detected.
-U, --unsafe        Start database unsafely, skipping recovery. Use under
                    support guidance only.
```

Usage: stop_db [options]

Options:

```
-h, --help          show this help message and exit
-d DB, --database=DB  Name of database to be stopped
-p DBPASSWORD, --password=DBPASSWORD
                    Database password in single quotes
-F, --force         Force the databases to shutdown, even if users are
                    connected.
--timeout=NONINTERACTIVE_TIMEOUT
                    set a timeout (in seconds) to wait for actions to
                    complete ('never') will wait forever (implicitly sets
                    -i)
-i, --noprompts     do not stop and wait for user input(default false)
```

Usage: stop_host [options]

Options:

```
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed using a SIGTERM signal
--compat21          (deprecated) Use Vertica 2.1 method using node names
                    instead of hostnames
```

Usage: stop_node [options]

Options:

```
-h, --help          show this help message and exit
-s HOSTS, --hosts=HOSTS
                    comma-separated list of hosts on which the vertica
                    process is to be killed using a SIGTERM signal
--compat21          (deprecated) Use Vertica 2.1 method using node names
                    instead of hostnames
```

Usage: uninstall_package [options]

Options:

```
-h, --help          show this help message and exit
-d DBNAME, --dbname=DBNAME
                    database name
-p PASSWORD, --password=PASSWORD
                    database admin password
-P PACKAGE, --package=PACKAGE
                    specify package or 'all' or 'default'
```

```
-----  
Usage: upgrade_license_key --database mydb --license my_license.key  
upgrade_license_key --install --license my_license.key  
  
Updates the vertica license.  
  
Without '--install', updates the license used by the database and  
the admintools license cache.  
  
With '--install', updates the license cache in admintools that  
is used for creating new databases.  
  
Options:  
-h, --help          show this help message and exit  
-d DB, --database=DB  Name of database. Cannot be used with --install.  
-l LICENSE, --license=LICENSE  
                    Required - path to the license.  
-i, --install        When option is included, command will only update the  
                    admintools license cache. Cannot be used with  
                    --database.  
-p PASSWORD, --password=PASSWORD  
                    Database password.  
-----  
Usage: view_cluster [options]  
  
Options:  
-h, --help          show this help message and exit  
-x, --xexpand        show the full cluster state, node by node  
-d DB, --database=DB  filter the output for a single database
```

Operating the Database

This topic explains how to start and stop your Vertica database, and how to use the database index tool:

- [Starting the Database](#)
- [Stopping the Database](#)
- [CRC and Sort Order Check](#)

Start the Database

You can start a database through one of the following:

- [Management Console](#)
- Administration Tools interface
- Command line

Administration Tools

1. Open the Administration Tools and select [View Database Cluster State](#) to make sure that all nodes are down and that no other database is running.
2. Open the Administration Tools. See [Using the Administration Tools](#) for information about accessing the Administration Tools.
3. On the **Main Menu**, select **Start Database**, and then select **OK**.
4. Select the database to start, and then click **OK**.

Caution: You should start only one database at a time. If you start more than one database at any time, the results can be unpredictable. Users might encounter resource conflicts or perform operations on the wrong database.

5. Enter the database password and click **OK**.

6. When prompted that the database started successfully, click **OK**.
7. Check the log files to make sure that no startup problems occurred.

Command Line

You can start a database with the [command line tool](#) `start_db`:

```
$ /opt/vertica/bin/admintools -t start_db -d db-name  
[-p password] [-F]
```

Option	Description
<code>-p</code> <code>--password</code>	Required only during database creation, when you install a new license. If the license is valid, the option <code>-p</code> (or <code>--password</code>) is not required to start the database and is silently ignored. This is by design, as the database can only be started by the user who (as part of the <code>verticadba</code> UNIX user group) initially created the database or who has root or su privileges. If the license is invalid, Vertica uses the <code>-p password</code> argument to attempt to upgrade the license with the license file stored in <code>/opt/vertica/config/share/license.key</code> .
<code>-F</code> <code>--force</code>	Forces the database to start at an epoch that precedes detection of data consistency problems

Following is an example of using `start_db` on a standalone node:

```
$ /opt/vertica/bin/admintools -t start_db -d VMart  
Info:  
no password specified, using none  
Node Status: v_vmart_node0001: (DOWN)  
Node Status: v_vmart_node0001: (UP)  
Database VMart started successfully
```

Stopping the Database

There are many occasions when you must stop a database, for example, before an upgrade or performing various maintenance tasks. You can stop a running database through one of the following:

- [Management Console](#)
- [Administration Tools interface](#)
- [Command line](#)

You cannot stop a running database if any users are connected or Database Designer is building or deploying a database design.

Note: If the Tuple Mover is doing a Moveout operation then the database cannot stop until the Moveout is complete. If the database is not stopping after you issue a stop command then you can verify a Moveout operation is preventing the database from stopping by looking at the Vertica log file. See [Monitoring Log Files](#) for details on locating and viewing the Vertica log. Tuple Mover operations that are in progress display an INFO message: [Session] <INFO> closeAndWaitAllSessions: waiting for session to end. The database will stop after the Moveout completes, you do not need to take additional action other than waiting.

Administration Tools

To stop a running database with admintools:

1. [Verify that all cluster nodes are up](#). If any nodes are down, [identify and restart them](#).
2. Close all user sessions:
 - Identify all users with active sessions by querying the [SESSIONS](#) system table. Notify users of the impending shutdown and request them to shut down their sessions.
 - Prevent users from starting new sessions by temporarily resetting configuration parameter [MaxClientSessions](#) to 0:

```
=> ALTER DATABASE mydb SET MaxClientSessions = 0;
```

- Close all remaining user sessions with Vertica functions [CLOSE_SESSION](#) and [CLOSE_ALL_SESSIONS](#).

Note: You can also force a database shutdown and block new sessions with the function [SHUTDOWN](#).

3. Open Vertica [Administration Tools](#).
4. From the Main Menu:
 - Select Stop Database
 - Click **OK**
5. Select the database to stop and click **OK**.
6. Enter the password (if asked) and click **OK**.
7. When prompted that database shutdown is complete, click **OK**.

Command Line

You can stop a database with the [command line tool](#) `stop_db`:

```
$ /opt/vertica/bin/admintools -t stop_db -d db-name  
[-p password] [-F]
```

Use the option `-F` (or `--force`) to override all user connections and force a shutdown.

CRC and Sort Order Check

As a superuser, you can run the Index tool on a Vertica database to perform two tasks:

- Run a per-block cyclic redundancy check (CRC) on data storage to verify data integrity.
- Check that the sort order in ROS containers is correct.

If the database is down, invoke the Index tool from the Linux command line. If the database is up, invoke it as a SQL statement from vsql:

Operation	Database Down	Database Up
Run CRC	<code>/opt/vertica/bin/vertica -D catalog-path -v</code>	<code>select run_index_tool ('checkcrc'); select run_index_tool ('checkcrc', 'true');</code>
Check sort order	<code>/opt/vertica/bin/vertica -D catalog-path -I</code>	<code>select run_index_tool ('checksort'); select run_index_tool ('checksort', 'true');</code>

If you run the Index tool in vsql as a SQL statement, you can specify that it analyze all cluster nodes by setting the optional Boolean parameter to `true` (1). If this parameter is omitted, the Index tool runs only on the current node.

If invoked from the command line, the Index tool runs only on the current node. However, the Index tool can run on multiple nodes simultaneously. Invoke the Index tool binary from the `/opt/vertica/bin` directory.

Viewing Results

The Index tool writes summary information about its operation to standard output; detailed information on results is logged in one of two locations, depending on the environment where you invoke the tool:

- **Invoked from the command-line:** Results written to `indextool.log` file in the database catalog directory.
- **Invoked from vsql:** Results written to `vertica.log` on the current node.

Privileges

Restricted to superusers.

Running a Cyclic Redundancy Check

The Index tool can run a cyclic redundancy check (CRC) on each block of existing data storage to check the data integrity of ROS data blocks.

Running the Tool

You can invoke the Index tool from the command line or from `vsql`, depending on whether the database is up or down:

- **If the database is down:**

Invoke the Index tool from the Linux command line. For example:

```
dbadmin@localhost bin]$ /opt/vertica/bin/vertica -D /home/dbadmin/VMart/v_vmart_node0001_catalog -v
```

The Index tool writes summary information about its operation to standard output, and logs detailed information in `indextool.log` in the database catalog directory.

- **If the database is up:**

Invoke the Index tool it as a SQL statement from `vsql` with the argument `checkcrc`. To run the index tool on all nodes, also set the tool's optional Boolean parameter to `true`. If this parameter is omitted, the Index tool runs only on the current node. For example, the following SQL statement runs a CRC on all cluster nodes:

```
select run_index_tool ('checkcrc', 'true');
```

The Index tool writes summary information about its operation to standard output, and logs detailed information in `vertica.log` on the current node.

Handling CRC Errors

Vertica evaluates the CRC values in each ROS data block each time it fetches data disk to process a query. If CRC errors occur while fetching data, the following information is written to the `vertica.log` file:

```
CRC Check Failure Details:File Name:  
File Offset:  
Compressed size in file:  
Memory Address of Read Buffer:  
Pointer to Compressed Data:  
Memory Contents:
```

The Event Manager is also notified of CRC errors, so you can use an SNMP trap to capture CRC errors:

```
"CRC mismatch detected on file <file_path>. File may be corrupted. Please check hardware and drivers."
```

If you run a query from `vsq`, ODBC, or JDBC, the query returns a `FileColumnReader ERROR`. This message indicates that a specific block's CRC does not match a given record as follows:

```
hint: Data file may be corrupt. Ensure that all hardware (disk and memory) is working properly.  
Possible solutions are to delete the file <pathname> while the node is down, and then allow the node  
to recover, or truncate the table data.code: ERRCODE_DATA_CORRUPTED
```

Checking Sort Order

If ROS data is not sorted correctly in the projection's order, query results that rely on sorted data will be incorrect. You can use the Index tool to check the ROS sort order if you suspect or detect incorrect query results. The Index tool evaluates each ROS row to determine whether it is sorted correctly. If the check locates a row that is not in order, it writes an error message to the log file with the row number and contents of the unsorted row.

Running the Tool

You can invoke the Index tool from the command line or from `vsq`, depending on whether the database is up or down:

- **If the database is down:**

Invoke the Index tool from the Linux command line. For example:

```
$ /opt/vertica/bin/vertica -D /home/dbadmin/VMart/v_vmart_node0001_catalog -I
```

The Index tool writes summary information about its operation to standard output, and logs detailed information in `indextool.log` in the database catalog directory.

- **If the database is up:**

Invoke the Index tool from `vsq` as a SQL statement with the argument `checksort`. To run the index tool on all nodes, also set the tool's optional Boolean parameter to `true`. If this parameter is omitted, the Index tool runs only on the current node.

For example, the following SQL statement runs a CRC on all cluster nodes:

```
select run_index_tool ('checksort', 'true');
```

The Index tool writes summary information about its operation to standard output, and logs detailed information in `vertica.log` on the current node.

Reviewing Errors

1. Open the `indextool.log` file. For example:

```
$ cd VMart/v_check_node0001_catalog
```

2. Look for error messages that include an OID number and the string `Sort Order Violation`. For example:

```
<INFO> ...on oid 45035996273723545: Sort Order Violation:
```

3. Find detailed information about the sort order violation string by running `grep` on `indextool.log`. For example, the following command returns the line before each string (`-B1`), and the four lines that follow (`-A4`):

```
[15:07:55][vertica-s1]: grep -B1 -A4 'Sort Order Violation:' /my_host/databases/check/v_check_node0001_catalog/indextool.log  
2012-06-14 14:07:13.686 unknown:0x7fe1da7a1950 [EE] <INFO> An error occurred when running index tool thread on oid 45035996273723537:
```

```
Sort Order Violation:
Row Position: 624
Column Index: 0
Last Row: 2576000
This Row: 2575000
--
2012-06-14 14:07:13.687 unknown:0x7fe1dafa2950 [EE] <INFO> An error occurred when running index
tool thread on oid 45035996273723545:
Sort Order Violation:
Row Position: 3
Column Index: 0
Last Row: 4
This Row: 2
--
```

4. Find the projection where a sort order violation occurred by querying the `storage_containers` system table. Use a `storage_oid` equal to the OID value listed in `indextool.log`. For example:

```
=> select * from storage_containers where storage_oid = 45035996273723545;
```

Managing Tables

You can create two types of tables in Vertica, columnar and flexible. You can create both types as persistent or temporary. You can also create views that query a specific set of table columns.

Creating Tables

CREATE TABLE creates a table in the Vertica logical schema. For example:

```
CREATE TABLE vendor_dimension (  
  vendor_key      INTEGER      NOT NULL PRIMARY KEY,  
  vendor_name     VARCHAR(64),  
  vendor_address  VARCHAR(64),  
  vendor_city     VARCHAR(64),  
  vendor_state    CHAR(2),  
  vendor_region   VARCHAR(32),  
  deal_size       INTEGER,  
  last_deal_update DATE  
);
```

Table Data Storage

Unlike traditional databases that store data in tables, Vertica physically stores table data in projections, which are collections of table columns. Projections store data in a format that optimizes query execution. Similar to materialized views, they store result sets on disk rather than compute them each time they are used in a query.

In order to query or perform any operation on a Vertica table, the table must have one or more projections associated with it. For more information, see [Physical Schema](#) in Vertica Concepts.

See Also

- [Altering Table Definitions](#)
- [Creating Temporary Tables](#)
- [Creating a Table from Other Tables](#)
- [Creating External Tables](#)

Creating Temporary Tables

`CREATE TEMPORARY TABLE` creates a table whose data persists only during the current session. Temporary table data is never visible to other sessions.

By default, all temporary table data is transaction-scoped—that is, the data is discarded when a `COMMIT` statement ends the current transaction. If `CREATE TEMPORARY TABLE` includes the parameter `ON COMMIT PRESERVE ROWS`, table data is retained until the current session ends.

Temporary tables can be used to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated—for example, the tool first gets a result set, then queries the result set, and so on.

When you create a temporary table, Vertica automatically generates a default projection for it. For more information, see [Auto-Projections](#).

Global versus Local Tables

`CREATE TEMPORARY TABLE` can create tables at two scopes, global and local, through the keywords `GLOBAL` and `LOCAL`, respectively:

Global temporary tables	Vertica creates global temporary tables in the public schema. Definitions of these tables are visible to all sessions, and persist across sessions until they are explicitly dropped. Multiple users can access the table concurrently. Table data is session-scoped, so it is visible only to the session user, and is discarded when the session ends.
Local temporary tables	Vertica creates local temporary tables in the <code>V_TEMP_SCHEMA</code> namespace and inserts them transparently into the user's search path. These tables are visible only to the session where they are created. When the session ends, Vertica automatically drops the table and its data.

Data Retention

You can specify whether temporary table data is transaction- or session-scoped:

- **ON COMMIT DELETE ROWS** (default): Vertica automatically removes all table data when each transaction ends.
- **ON COMMIT PRESERVE ROWS**: Vertica preserves table data across transactions in the current session. Vertica automatically truncates the table when the session ends.

Note: If you create a temporary table with `ON COMMIT PRESERVE ROWS`, you cannot add projections for that table if it contains data. You must first remove all data from that table with `TRUNCATE TABLE`.

You can create projections for temporary tables created with `ON COMMIT DELETE ROWS`, whether populated with data or not. However, `CREATE PROJECTION` ends any transaction where you might have added data, so projections are always empty.

ON COMMIT DELETE ROWS

By default, Vertica removes all data from a temporary table, whether global or local, when the current transaction ends.

For example:

```
=> CREATE TEMPORARY TABLE tempDelete (a int, b int);
CREATE TABLE
=> INSERT INTO tempDelete VALUES(1,2);
OUTPUT
-----
      1
(1 row)

=> SELECT * FROM tempDelete;
 a | b
---+---
 1 | 2
(1 row)

=> COMMIT;
COMMIT

=> SELECT * FROM tempDelete;
 a | b
---+---
(0 rows)
```

If desired, you can use `DELETE` within the same transaction multiple times, in order to refresh table data repeatedly.

ON COMMIT PRESERVE ROWS

You can specify that a temporary table retain data across transactions in the current session, by defining the table with the keywords `ON COMMIT PRESERVE ROWS`. Vertica automatically removes all data from the table only when the current session ends.

For example:

```
=> CREATE TEMPORARY TABLE tempPreserve (a int, b int) ON COMMIT PRESERVE ROWS;
CREATE TABLE
=> INSERT INTO tempPreserve VALUES (1,2);
  OUTPUT
-----
      1
(1 row)

VMart=> COMMIT;
COMMIT
VMart=> SELECT * FROM tempPreserve;
  a | b
---+---
  1 | 2
(1 row)

=> INSERT INTO tempPreserve VALUES (3,4);
  OUTPUT
-----
      1
(1 row)

VMart=> COMMIT;
COMMIT
VMart=> SELECT * FROM tempPreserve;
  a | b
---+---
  1 | 2
  3 | 4
(2 rows)
```

Creating a Table from Other Tables

You can create a table from other tables in two ways:

- [Replicate an existing table](#) through `CREATE TABLE...LIKE`.
- [Create a table from a query](#) through `CREATE TABLE...AS`.

Replicating a Table

You can create a table from an existing one using `CREATE TABLE` with the `LIKE` clause:

```
CREATE TABLE [schema.]table-name LIKE [schema.]existing-table
...[ {INCLUDING | EXCLUDING} PROJECTIONS ]
...[ Load-method ]
...[ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

Creating a table with LIKE replicates the source table definition and any storage policy associated with it. CREATE TABLE . . . LIKE copies all table constraints except foreign key constraints. It does not copy table data or expressions on columns and constraints.

Including Projections

You can qualify the LIKE clause with INCLUDING PROJECTIONS or EXCLUDING PROJECTIONS, which specify whether to copy projections from the source table:

- EXCLUDING PROJECTIONS (default): Do not copy projections from the source table.
- INCLUDING PROJECTIONS: Copy current projections from the source table. INCLUDING PROJECTIONS also copies all table constraints except foreign key constraints. If the table is associated with a storage policy, the policy association is also replicated. Vertica names the new projections according to Vertica [naming conventions](#), which avoids name conflicts with existing objects.

Specifying a Load Method

You can qualify the LIKE clause with a load method, one of the following:

- AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads.
- DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads.
- TRICKLE: Loads data only into WOS, suitable for frequent incremental loads.

For details, see [Choosing a Load Method](#) in the Administrator's Guide.

Including Schema Privileges

You can specify default inheritance of schema privileges for the new table:

- EXCLUDE [SCHEMA] PRIVILEGES (default) disables inheritance of privileges from the schema
- INCLUDE [SCHEMA] PRIVILEGES grants the table the same privileges granted to its schema

For more information see [Grant Inherited Privileges](#).

Restrictions

The following restrictions apply to the source table:

- It cannot have out-of-date projections.
- It cannot be a temporary table.

Example

1. Create the table states:

```
=> CREATE TABLE states (  
    state char(2) NOT NULL, bird varchar(20), tree varchar (20), tax float, stateDate char  
(20))  
    PARTITION BY state;
```

2. Populate the table with data:

```
INSERT INTO states VALUES ('MA', 'chickadee', 'american_elm', 5.675, '07-04-1620');  
INSERT INTO states VALUES ('VT', 'Hermit_Thrasher', 'Sugar_Maple', 6.0, '07-04-1610');  
INSERT INTO states VALUES ('NH', 'Purple_Finch', 'White_Birch', 0, '07-04-1615');  
INSERT INTO states VALUES ('ME', 'Black_Cap_Chickadee', 'Pine_Tree', 5, '07-04-1615');  
INSERT INTO states VALUES ('CT', 'American_Robin', 'White_Oak', 6.35, '07-04-1618');  
INSERT INTO states VALUES ('RI', 'Rhode_Island_Red', 'Red_Maple', 5, '07-04-1619');
```

3. View the table contents:

```
=> SELECT * FROM states;
```

state	bird	tree	tax	stateDate
VT	Hermit_Thrasher	Sugar_Maple	6	07-04-1610
CT	American_Robin	White_Oak	6.35	07-04-1618
RI	Rhode_Island_Red	Red_Maple	5	07-04-1619
MA	chickadee	american_elm	5.675	07-04-1620
NH	Purple_Finch	White_Birch	0	07-04-1615
ME	Black_Cap_Chickadee	Pine_Tree	5	07-04-1615

(6 rows)

4. Create a sample projection and refresh:

See Also

- [Creating Tables](#)
- [Creating Temporary Tables](#)
- [Creating External Tables](#)
- [Creating a Table from a Query](#)

Creating a Table from a Query

CREATE TABLE can specify an AS clause, to create a table from a query, as follows:

```
CREATE [TEMPORARY] TABLE [schema.]table-name
... [ ( column-name-list ) ]
... [ Load-method ]
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
AS [ /*+ hint[, hint] */ ] [ AT epoch ] query [ ENCODED BY column-ref-list ]
```

Vertica creates a table from the query results and loads the result set into it. For example:

```
=> CREATE TABLE cust_basic_profile AS SELECT
    customer_key, customer_gender, customer_age, marital_status, annual_income, occupation
    FROM customer_dimension WHERE customer_age>18 AND customer_gender !='';
CREATE TABLE
=> SELECT customer_age, annual_income, occupation FROM cust_basic_profile
    WHERE customer_age > 23 ORDER BY customer_age;
customer_age | annual_income |      occupation
-----+-----+-----
      24 |      469210 | Hairdresser
      24 |      140833 | Butler
      24 |      558867 | Lumberjack
      24 |      529117 | Mechanic
      24 |      322062 | Acrobat
      24 |      213734 | Writer
      ...
```

AS Clause Options

You can qualify the AS clause with one or both of the following hints:

- A load method hint: [AUTO](#), [DIRECT](#), or [TRICKLE](#)

Note: The `CREATE TABLE` statement can also specify a load method. However, this load method applies to load operations only after the table is created.

- [LABEL](#): Assigns a label to a statement to identify it for profiling and debugging. See [Labeling the AS Clause](#) below.

You can also specify that the query return historical data with `AT epoch`, where *epoch* is one of the following:

- `EPOCH LATEST`: Return data from the latest committed DML transaction.
- `EPOCH integer`: Return data from the *integer*-specified epoch.
- `TIME 'timestamp'`: Return data from the *timestamp*-specified epoch.

Note: These options are invalid for external tables.

Labeling the AS Clause

You can embed a [LABEL](#) hint in an AS clause in two places:

- Immediately after the keyword `AS`:

```
CREATE TABLE myTable AS /*+LABEL myLabel*/...
```

- In the `SELECT` statement:

```
CREATE TABLE myTable AS SELECT /*+LABEL myLabel*/
```

If the AS clause contains a `LABEL` hint in both places, the first label has precedence.

Note: Labels are invalid for external tables.

Specifying a Load Method

You can qualify the AS clause with one of these load method hints:

- [AUTO](#)
- [DIRECT](#)
- [TRICKLE](#)

Vertica applies the load hint to the data that is loaded into the new table. You can also specify a load method in the `CREATE TABLE` statement, which is saved to the table schema. However, the load method applies only to load operations after the table is created.

In the following example, table `bar` is created and loaded from table `foo` with a `CREATE TABLE AS` statement. The `CREATE` statement is qualified with the load method `DIRECT`, which is saved to `bar`'s schema definition. The load method specifies that all future load operations are written to ROS. The `CREATE` statement's `AS` clause also contains the load method hint `/*+DIRECT*/`, which specifies to load the data queried from `foo` into ROS:

```
=> SELECT * FROM foo;
 col1 | col2 | col3
-----+-----
    4 |    5 |    6
    1 |    2 |    3
(2 rows)

=> CREATE TABLE bar DIRECT AS SELECT /*+DIRECT*/ * FROM foo;
CREATE TABLE
dbadmin=> SELECT EXPORT_TABLES ('', 'bar');
                                EXPORT_TABLES
-----
CREATE TABLE public.bar
(
  col1 int,
  col2 int,
  col3 int
) DIRECT ;
(1 row)
```

For details, see [Choosing a Load Method](#).

Note: Load methods cannot be specified for external tables.

Loading Historical Data

You can qualify a `CREATE TABLE AS` query with the option, with `AT epoch`. `AT epoch` specifies to return historical data, where `epoch` is one of the following:

- `EPOCH LATEST`: Return data from the latest committed DML transaction.
- `EPOCH integer`: Return data from the `integer`-specified epoch.

- `TIME 'timestamp'`: Return data from the *timestamp*-specified epoch.

For details, see [Historical Queries](#) in Analyzing Data.

Note: This option is invalid for external tables.

Zero-Width Column Handling

If the query returns a column with zero width, Vertica automatically converts it to a `VARCHAR(80)` column. For example:

```
=> CREATE TABLE example AS SELECT '' AS X;
CREATE TABLE
=> SELECT EXPORT_TABLES ('', 'example');
          EXPORT_TABLES
-----
CREATE TEMPORARY TABLE public.example
(
  X varchar(80)
);
```

Requirements and Restrictions

- If you create a temporary table from a query, you must specify `ON COMMIT PRESERVE ROWS` in order to load the result set into the table. Otherwise, Vertica creates an empty table.
- If the query output has expressions other than simple columns, such as constants or functions, you must specify an alias for that expression, or list all columns in the column name list.

See Also

- [Creating Tables](#)
- [Creating Temporary Tables](#)
- [Creating External Tables](#)
- [Replicating a Table](#)

Creating External Tables

You create an external table using the `CREATE EXTERNAL TABLE AS COPY` statement. You cannot create temporary external tables. For the syntax details to create an external table, see the [CREATE EXTERNAL TABLE](#) statement in the SQL Reference Manual.

Note: Each table can have a maximum of 1600 columns.

Required Permissions for External Tables

You must be a database superuser to create external tables.

Permission requirements to use (`SELECT` from) external tables differ from those of other tables. By default, once external tables exist, you must also be a database superuser to access them through a `SELECT` statement.

To allow users without superuser access to query external tables, an administrator must create a 'user' storage location and grant those users read access to the location. See [CREATE LOCATION](#), and [GRANT \(Storage Location\)](#). This location must be a parent of the path used in the `COPY` statement when creating the external table.

COPY Statement Definition

When you create an external table, table data is not added to the database, and no projections are created. Instead, Vertica performs a syntactic check of the `CREATE EXTERNAL TABLE . . .` statement, and stores the table name and `COPY` statement definition in the catalog. When a `SELECT` query references an external table, Vertica parses and executes the stored `COPY` statement to obtain the referenced data. Successfully returning data from an external table requires that the `COPY` definition be correct, and that other dependencies, such as files, nodes, and other resources are accessible and available at query-time.

If the maximum length of a column is smaller than the actual data, such as a `VARCHAR` that is too short, Vertica truncates the data and logs the event.

When using the [COPY parameter](#) on any node, confirm that the source file definition is identical on all nodes. Specifying different external files can produce inconsistent results.

For more information about checking the validity of the external table `COPY` definition, see [Validating External Tables](#).

NOT NULL Constraints

Do not specify a NOT NULL column constraint, unless you are certain that the external data does not contain NULL values. Otherwise, you may see unexpected query results. For example, a SELECT statement for an external table with a NOT NULL constraint will reject a column value if it is not NULL.

Canceling the Create Query

Canceling a CREATE EXTERNAL TABLE AS COPY statement can cause unpredictable results. If you enter a query to create an external table, and it is incorrect (for example, you inadvertently specify the wrong external location), wait for the query to complete. When the external table exists, use [DROP TABLEs](#) definition.

Developing User-Defined Load (UDL) Functions for External Tables

You can create external tables with your own load functions. For more information about developing user-defined load functions, see [User Defined Load \(UDL\)](#) and the extended [COPY](#) syntax in the SQL Reference Manual.

Examples

Examples of external table definitions:

```
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat' DELIMITER ',';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM 'hdfs:///dat/ext1.dat';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat.bz2' BZIP DELIMITER ',';
=> CREATE EXTERNAL TABLE ext1 (x integer, y integer) AS COPY (x as '5', y) FROM '/tmp/ext1.dat.bz2'
BZIP DELIMITER ',';
```

To allow users without superuser access to use these tables, create a location for 'user' usage and grant access to it. This example shows granting access to a user named Bob to any external table whose data is located under /tmp (including in subdirectories to any depth):

```
=> CREATE LOCATION '/tmp' ALL NODES USAGE 'user';
=> GRANT ALL ON LOCATION '/tmp' to Bob;
```

See Also

- [COPY](#)
- [CREATE EXTERNAL TABLE AS COPY](#)

Validating External Tables

When you create an external table, Vertica validates the syntax of the `CREATE EXTERNAL TABLE AS COPY FROM` statement. For example, if you omit a required keyword in the statement (such as `FROM`), creating the external table fails:

```
VMart=> create external table ext (ts timestamp,d varchar) as copy '/home/dbadmin/designer.log';  
ERROR 2778: COPY requires a data source; either a FROM clause or a WITH SOURCE for a user-defined  
source
```

Checking other components of the `COPY` definition (such as path statements and node availability) does not occur until a `SELECT` query references the external table.

To validate an external table definition, run a `SELECT` query that references the external table. Check that the returned query data is what you expect. If the query does not return data correctly, check the `COPY` exception and rejected data log files.

Since the `COPY` definition determines what occurs when you query an external table, `COPY` statement errors can reveal underlying problems. For more information about `COPY` exceptions and rejections, see [Capturing Load Rejections and Exceptions](#).

Setting Maximum Exceptions

Querying external table data with an incorrect `COPY FROM` statement definition can potentially result in many rejected rows. To limit the number of rejections, Vertica sets the maximum number of retained rejections with the `ExternalTablesExceptionsLimit` configuration parameter. The default value is 100. Setting the `ExternalTablesExceptionsLimit` to `-1` removes the limit, but is not recommended.

If `COPY` errors reach the maximum number of rejections, the external table query continues, but `COPY` generates a warning in the `vertica.log`, and does not report subsequent rejected rows.

Note: Using the `ExternalTablesExceptionsLimit` configuration parameter differs from the `COPY` statement `REJECTMAX` parameter. The `REJECTMAX` value controls how many rejected rows to permit before causing the load to fail. If `COPY` encounters a number of rejected rows equal to or greater than `REJECTMAX`, `COPY` aborts execution. A `vertica.log` warning is not generated if `COPY` exceeds `REJECTMAX`.

Working with External Tables

After creating external tables, you access them as any other table. However, you cannot perform `UPDATE`, `INSERT`, or `DELETE` operations on external tables.

Managing Resources for External Tables

External tables require minimal additional resources. When you use a select query for an external table, Vertica uses a small amount of memory when reading external table data, since the table contents are not part of your database and are parsed each time the external table is used.

Vertica Does Not Back Up External Tables

Since the data in external tables is managed outside of Vertica, only the external table definitions, not the data files, are included in database backups. Arrange for a separate backup process for your external table data.

Using Sequences and Identity Columns in External Tables

The `COPY` statement definition for external tables can include identity columns and sequences. Whenever a select statement queries the external table, sequences and identity columns are re-evaluated. This results in changing the external table column values, even if the underlying external table data remains the same.

Viewing External Table Definitions

When you create an external table, Vertica stores the `COPY` definition statement in the `table_definition` column of the `v_catalog.tables` system table.

1. To list all tables, use a `select * query`, as shown:

```
select * from v_catalog.tables where table_definition <> '';
```

2. Use a query such as the following to list the external table definitions (`table_definition`):

```
select table_name, table_definition from v_catalog.tables;
table_name | table_definition
-----+-----
t1         | COPY          FROM 'TMPDIR/external_table.dat' DELIMITER ','
t1_copy   | COPY          FROM 'TMPDIR/external_table.dat' DELIMITER ','
t2         | COPY FROM 'TMPDIR/external_table2.dat' DELIMITER ','
(3 rows)
```

External Table DML Support

Following are examples of supported queries, and others that are not:

Supported	Unsupported
<code>SELECT * FROM external_table;</code>	<code>DELETE FROM external_table WHERE x = 5;</code>
<code>SELECT * FROM external_table where col1=4;</code>	<code>INSERT INTO external_table SELECT * FROM ext;</code>
<code>DELETE FROM internal_table WHERE id IN (SELECT x FROM external_table);</code>	
<code>INSERT INTO internal_table SELECT * FROM external_table;</code>	<code>SELECT * FROM external_table FOR UPDATE;</code>

Using External Table Values

Following is a basic example of how you could use the values of an external table.

1. Create and display the contents of a file with some integer values:

```
[dbadmin@localhost ~]$ more ext.dat1
2
3
4
5
6
```

```
7  
8  
10  
11  
12
```

2. Create an external table pointing at ext.dat :

```
VMart=> create external table ext (x integer) as copy from '/home/dbadmin/ext.dat';  
CREATE TABLE
```

3. Select the table contents:

```
VMart=> select * from ext;  
x  
----  
1  
2  
3  
4  
5  
6  
7  
8  
10  
11  
12  
(11 rows)
```

4. Perform evaluation on some external table contents:

```
VMart=> select ext.x, ext.x + ext.x as double_x from ext where x > 5;  
x | double_x  
----+-----  
6 |      12  
7 |      14  
8 |      16  
10 |     20  
11 |     22  
12 |     24  
(6 rows)
```

5. Create a second table (second), also with integer values:

```
VMart=> create table second (y integer);  
CREATE TABLE
```

6. Populate the table with some values:

```
VMart=> copy second from stdin;Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 1  
>> 1  
>> 3  
>> 4  
>> 5  
>> \.
```

7. Join the external table (ext) with the table created in Vertica, called second:

```
VMart=> select * from ext join second on x=y;  
 x | y  
----  
 1 | 1  
 1 | 1  
 3 | 3  
 4 | 4  
 5 | 5  
(5 rows)
```

Using External Tables

External tables let you query data stored in files accessible to the Vertica database, but not managed by it. Creating external tables supplies read-only access through SELECT queries. You cannot modify external tables through DML commands, such as INSERT, UPDATE, DELETE, and MERGE.

Using CREATE EXTERNAL TABLE AS COPY Statement

You create external tables with the CREATE EXTERNAL TABLE AS COPY... statement, shown in this basic example:

```
CREATE EXTERNAL TABLE tbl(i INT) AS COPY (i) FROM 'path1' ON v_vmart_node0001, 'path2' ON v_vmart_  
node0002;
```

For more details on the supported options to create an external table, see the [CREATE EXTERNAL TABLE](#) statement in the SQL Reference Manual.

The data you specify in the FROM clause of a CREATE EXTERNAL TABLE AS COPY statement can reside in one or more files or directories, and on one or more nodes. After successfully creating an external table, Vertica stores the table name and its COPY definition. Each time a select query references the external table, Vertica parses the COPY statement definition again to access the data. Here is a sample select statement:

```
SELECT * FROM tbl WHERE i > 10;
```

Storing Vertica Data in External Tables

While there are many requirements for you to use external table data, one reason is to store infrequently-accessed Vertica data on low-cost external media. If external storage is a goal at your site, the process to accomplish that requires exporting the older data to a text file, creating a bzip or gzip file of the export data, and saving the compressed file on an NFS disk. You can then create an external table to access the data any time it is required.

Calculating Exact Row Count for External Tables

To calculate the exact number of rows in an external table, use [ANALYZE_EXTERNAL_ROW_COUNT](#). The Optimizer uses this count to optimize for queries that access external tables.

In particular, if an external table participates in a join, the Optimizer can now identify the smaller table to be used as the inner input to the join, resulting in better query performance.

Using External Tables with User-Defined Load (UDL) Functions

You can also use external tables in conjunction with the UDL functions that you create. For more information about using UDLs, see [User Defined Load \(UDL\)](#) in Extending Vertica.

Organizing External Table Data

If the data you store in external tables changes regularly (for instance, each month in the case of storing recent historical data), your COPY definition statement can use wildcards to make parsing the stored COPY statement definition more dynamic. For instance, if you store monthly data on an NFS mount, you could organize monthly files within a top-level directory for a calendar year, such as:

```
/2012/monthly_archived_data/
```

In this case, the external table COPY statement will include a wildcard definition such as the following:

```
CREATE TABLE archive_data (...) AS COPY FROM '/nfs_name/2012/monthly_archived_data/*'
```

Whenever a Vertica query references the external table `months`, and Vertica parses the `COPY` statement, all stored data tables in the top-level `monthly_archived_data` directory are made accessible to the query.

Managing Table Columns

After you define a table, you can use `ALTER TABLE` to modify existing table columns. You can perform the following operations on a column:

- [Rename it](#).
- [Change its data type](#).
- [Set its default value](#).
- [Add](#) and [remove](#) constraints.

Renaming Columns

You rename a column with `ALTER TABLE` as follows:

```
ALTER TABLE [schema.]table-name RENAME [ COLUMN ] column-name TO new-column-name
```

The following example renames a column in the `Retail.Product_Dimension` table from `Product_description` to `Item_description`:

```
=> ALTER TABLE Retail.Product_Dimension  
    RENAME COLUMN Product_description TO Item_description;
```

If you rename a column that is referenced by a view, the column does not appear in the result set of the view even if the view uses the wild card (*) to represent all columns in the table. Recreate the view to incorporate the column's new name.

Changing a Column Data Type

You can change a table column's data type with `ALTER TABLE` as long as the change complies with the requirements and restrictions cited below.

Supported Data Type Conversions

You can change a column's data type if doing so does not require storage reorganization. After you modify a column's data type, data that you load conforms to the new definition.

Vertica supports conversion between the following data types:

Data Types	Notes
Binary types	Expansion and contraction allowed. Conversion not allowed between BINARY and VARBINARY types.
Character types	All conversions allowed, including between CHAR and VARCHAR
Exact numeric types	INTEGER, INT, BIGINT, TINYINT, INT8, SMALLINT, and all NUMERIC values of scale ≤ 18 and precision 0 are interchangeable. For NUMERIC data types, you cannot alter scale, but you can change the precision in the ranges (0-18), (19-37), and so on.

You can expand columns within the same class of data type. This is useful for storing longer strings in a column. Vertica validates the data before it performs the conversion. In general, you can also reduce column widths within the data type class, as long as existing column data is no greater than the new width. For details, see [Reducing Column Width](#).

Unsupported Data Type Conversions

Vertica does not allow data type conversion on types that require storage reorganization:

- Boolean type conversion to other types
- DATE/TIME type conversion
- Approximate numeric type conversions
- Between BINARY and VARBINARY types

You also cannot change a column's data type if the column is one of the following:

- Primary key
- Foreign key

- Included in the SEGMENTED BY clause of any projection for that table.

You can work around some of these restrictions. For details, see [Working With Column Data Conversions](#).

Reducing Column Width

In general, Vertica supports reductions to column widths within the data type class, as long as existing column data is no greater than the new width. Otherwise, Vertica returns an error and the conversion fails. For example, if you try to convert a column from `varchar(25)` to `varchar(10)` Vertica allows the conversion as long as all column data is no more than 10 characters.

In the following example, columns `y` and `z` are initially defined as `VARCHAR` data types, and loaded with values `12345` and `654321`, respectively. The attempt to reduce column `z`'s width to 5 fails because it contains six-character data. The attempt to reduce column `y`'s width to 5 succeeds because its content conforms with the new width:

```
=> CREATE TABLE t (x int, y VARCHAR, z VARCHAR);
CREATE TABLE
=> CREATE PROJECTION t_p1 AS SELECT * FROM t SEGMENTED BY hash(x) ALL NODES;
CREATE PROJECTION
=> INSERT INTO t values(1,'12345','654321');
OUTPUT
-----
      1
(1 row)

=> SELECT * FROM t;
 x |  y  |  z
---+-----+-----
  1 | 12345 | 654321
(1 row)

=> ALTER TABLE t ALTER COLUMN z SET DATA TYPE char(5);
ROLLBACK 2378: Cannot convert column "z" to type "char(5)"
HINT: Verify that the data in the column conforms to the new type
=> ALTER TABLE t ALTER COLUMN y SET DATA TYPE char(5);
ALTER TABLE
```

Purging Historical Data

You cannot reduce a column's width if Vertica retains any historical data that exceeds the new width. To reduce the column width, first remove that data from the table:

1. Advance the AHM to an epoch more recent than the historical data that needs to be removed from the table.
2. Purge the table of all historical data that precedes the AHM with the function `PURGE_TABLE`.

For example, given the previous example, you can update the data in column `t.z` as follows:

```
=> UPDATE t SET z = '54321';
OUTPUT
-----
      1
(1 row)

=> SELECT * FROM t;
 x | y | z
-----+-----
 1 | 12345 | 54321
(1 row)
```

Although no data in column `z` now exceeds 5 characters, Vertica retains the history of its earlier data, so attempts to reduce the column width to 5 return an error:

```
=> ALTER TABLE t ALTER COLUMN z SET DATA TYPE char(5);
ROLLBACK 2378: Cannot convert column "z" to type "char(5)"
HINT: Verify that the data in the column conforms to the new type
```

You can reduce the column width by purging the table's historical data as follows:

```
=> SELECT MAKE_AHM_NOW();
      MAKE_AHM_NOW
-----
AHM set (New AHM Epoch: 6350)
(1 row)

=> SELECT PURGE_TABLE('t');
                                     PURGE_TABLE
-----
Task: purge operation
(Table: public.t) (Projection: public.t_p1_b0)
(Table: public.t) (Projection: public.t_p1_b1)

(1 row)

=> ALTER TABLE t ALTER COLUMN z SET DATA TYPE char(5);
ALTER TABLE
```

Working With Column Data Conversions

Vertica conforms to the SQL standard by disallowing certain data conversions for table columns. However, you sometimes need to work around this restriction when you convert data

from a non-SQL database. The following examples describe one such workaround, using the following table:

```
=> CREATE TABLE sales(id INT, price VARCHAR) UNSEGMENTED ALL NODES;
CREATE TABLE
=> INSERT INTO sales VALUES (1, '$50.00');
OUTPUT
-----
      1
(1 row)

=> INSERT INTO sales VALUES (2, '$100.00');
OUTPUT
-----
      1
(1 row)

=> COMMIT;
COMMIT

=> SELECT * FROM SALES;
 id | price
-----+-----
  1 | $50.00
  2 | $100.00
(2 rows)
```

To convert the price column's existing data type from VARCHAR to NUMERIC, complete these steps:

1. [Add a new column for temporary use](#). Assign the column a NUMERIC data type, and derive its default value from the existing price column.
2. [Drop the original price column](#).
3. [Rename the new column to the original column](#).

Add a new column for temporary use

1. Add a column `temp_price` to table `sales`. You can use the new column temporarily, setting its data type to what you want (NUMERIC), and deriving its default value from the `price` column. Cast the default value for the new column to a NUMERIC data type and query the table:

```
=> ALTER TABLE sales ADD COLUMN temp_price NUMERIC(10,2) DEFAULT
SUBSTR(sales.price, 2)::NUMERIC;
ALTER TABLE

=> SELECT * FROM SALES;
 id | price | temp_price
```

```
-----+-----+-----  
1 | $50.00 | 50.00  
2 | $100.00 | 100.00  
(2 rows)
```

2. Use `ALTER TABLE` to drop the default expression from the new column `temp_price`. Vertica retains the values stored in this column:

```
=> ALTER TABLE sales ALTER COLUMN temp_price DROP DEFAULT;  
ALTER TABLE
```

Drop the original price column

Drop the extraneous `price` column. Before doing so, you must first advance the AHM to purge historical data that would otherwise prevent the drop operation:

1. Advance the AHM:

```
=> SELECT MAKE_AHM_NOW();  
MAKE_AHM_NOW  
-----  
AHM set (New AHM Epoch: 6354)  
(1 row)
```

2. Drop the original price column:

```
=> ALTER TABLE sales DROP COLUMN price CASCADE;  
ALTER COLUMN
```

Rename the new column to the original column

You can now rename the `temp_price` column to `price`:

1. Use `ALTER TABLE` to rename the column:

```
=> ALTER TABLE sales RENAME COLUMN temp_price to price;
```

2. Query the sales table again:

```
=> SELECT * FROM sales;  
id | price  
-----  
1 | 50.00
```

```
2 | 100.00  
(2 rows)
```

Changing Column Types in External Tables

Data from external tables is not stored in Vertica, so Vertica cannot validate any changes that you make in column data types. Vertica checks external table definitions only when it tries to read table data. If external column data is incompatible with the data type that is declared in the Vertica table, Vertica returns an error.

For example:

```
=> CREATE EXTERNAL TABLE t (a char(10), b binary(20)) AS COPY FROM '...';  
=> ALTER TABLE t ALTER COLUMN a SET DATA TYPE long varchar(1000000);  
ALTER TABLE  
=> ALTER TABLE t ALTER COLUMN b SET DATA TYPE long varbinary(1000000);  
ALTER TABLE
```

If you convert a column to a size that is too small for the data, Vertica truncates the data during the read. For example, if you convert a column from `varchar(25)` to `varchar(10)` and the column holds a 20-character string, Vertica reads the first ten and logs a truncation event.

Defining Column Values

You can define a column so Vertica automatically sets its value from an expression through one of the following clauses:

<code>DEFAULT <i>expression</i></code>	<p>Sets the default column values immediately in the following cases:</p> <ul style="list-style-type: none">• Adding a column with a DEFAULT expression to an existing table. Vertica populates the new column with its default values when it is added to the table. <p>Note: Altering an existing table column to specify a DEFAULT expression has no effect on existing values in that column. Vertica applies the DEFAULT expression only on new rows when they are added to the table, through load operations such as INSERT and COPY. To refresh the</p>
--	--

	<p>entire column with its DEFAULT expression, update the column as follows:</p> <pre>UPDATE table-name SET column-name=DEFAULT;</pre> <ul style="list-style-type: none">• Executing a table load operation such as INSERT. Vertica populates columns with default values in the new rows. Values in existing rows, including columns with DEFAULT expressions, remain unchanged.• Executing UPDATE on a table and setting default column to DEFAULT. See the example below.
<p>SET USING <i>expression</i></p>	<p>Sets the column value only when the function REFRESH_COLUMNS is invoked. This approach is especially useful for large denormalized (flattened) tables, where multiple columns get their values by querying other tables. For details, see Flattened Tables.</p> <p>In general, REFRESH_COLUMNS must be explicitly invoked by the user. One exception applies: when you use ALTER TABLE . . . ALTER COLUMN to apply SET USING to an existing column, or modify an existing SET USING expression. In this case, the DDL operation automatically invokes REFRESH_COLUMNS on the column, using UPDATE mode. After the refresh operation is complete, the DDL operation auto-commits the updates and returns. If the refresh operation fails, Vertica rolls back the entire DDL operation. Execution time can be significant if the refresh operation involves a large data set.</p>

Supported Expressions

DEFAULT and SET USING generally support the same expressions. These include:

- Queries (see [Flattened Tables](#))
- Other columns in the same table
- [Literals](#) (constants)
- All [operators](#) supported by Vertica

- The following categories of functions:
 - [Null-handling](#)
 - [User-defined scalar](#)
 - [System information](#)
 - [String](#)
 - [Mathematical](#)
 - [Formatting](#)

Expression Restrictions

The following restrictions apply to `DEFAULT` and `SET USING` expressions:

- The return value data type must match or be cast to the column data type.
- The expression must return a value that conforms to the column bounds. For example, a column that is defined as a `VARCHAR(1)` cannot be set to a default string of `abc`.
- The expression cannot specify correlated sub-queries.
- In a temporary table, `DEFAULT` and `SET USING` do not support sub-queries. If you try to create a temporary table with `DEFAULT` or `SET USING` using subquery expressions, Vertica returns an error.
- A column's `SET USING` expression cannot specify another column in the same table that also sets its value with `SET USING`. Similarly, a column's `DEFAULT` expression cannot specify another column in the same table that also sets its value with `DEFAULT`, or whose value is automatically set to a [sequence](#). However, a column's `SET USING` expression can specify another column that sets its value with `DEFAULT`.

Note: You can set a column's `DEFAULT` expression from another column in the same table that sets its value with `SET USING`. However, the `DEFAULT` column is typically set to `NULL`, as it is only set on load operations that initially set the `SET USING` column to `NULL`.

DEFAULT Restrictions

`DEFAULT` expressions cannot specify volatile functions with `ALTER TABLE...ADD COLUMN`. To specify volatile functions, use `CREATE TABLE` or `ALTER TABLE...ALTER COLUMN` statements.

SET USING Restrictions

The following restrictions apply to SET USING expressions:

- Volatile functions are not allowed.
- The expression cannot specify a sequence.
- You cannot modify SET USING data directly with DML operations such as COPY, INSERT, UPDATE, or MERGE. SET USING columns can only be updated by calling [REFRESH_COLUMNS](#). Attempts to modify SET USING columns return an error.
- Vertica limits the use of several meta-functions that copy table data: [COPY_TABLE](#), [COPY_PARTITIONS_TO_TABLE](#), [MOVE_PARTITIONS_TO_TABLE](#), and [SWAP_PARTITIONS_BETWEEN_TABLES](#). The following table describes these limitations:

SET USING columns in...	Limitation
Source and target table	All functions allowed only if each SET USING column in source table has an identical match in the target table
Source table only	<p>Allowed functions:</p> <ul style="list-style-type: none"> ■ COPY_TABLE ■ COPY_PARTITIONS_TO_TABLE ■ MOVE_PARTITIONS_TO_TABLE <p>Disallowed: SWAP_PARTITIONS_BETWEEN_TABLES</p> <p>The same set of restrictions apply if the target table is new.</p>
Target table only	No functions allowed.

Important: Several restrictions apply to Vertica's ability to refresh a SET USING column with [REFRESH_COLUMNS](#). For details, see [REFRESH_COLUMNS](#).

DEFAULT USING Columns and Restrictions

A column can specify both DEFAULT and SET USING constraints, as follows:

```
column-name data-type DEFAULT default-expr SET USING using-expr
```

Typically, both constraints specify the same expression. In this case, you can define the column as follows:

```
column-name data-type DEFAULT USING expression
```

DEFAULT USING columns support the same expressions as SET USING columns, and are subject to the same [restrictions](#).

Examples

Default column value derived from another column

1. Create a sample table called `t` with `timestamp`, `integer`, and `varchar(10)` columns:

```
=> CREATE TABLE t (a TIMESTAMP, b INT, c VARCHAR(10));
CREATE TABLE
=> INSERT INTO t VALUES ('2012-05-14 10:39:25', 2, 'MA');
OUTPUT
-----
      1
(1 row)
```

2. Use `ALTER TABLE` to add a fourth column that extracts the month from the timestamp value in column `a`:

```
=> ALTER TABLE t ADD COLUMN d INTEGER DEFAULT EXTRACT(MONTH FROM a);
ALTER TABLE
```

3. Verify this change:

```
select export_tables('', 't');
                                export_tables
-----
CREATE TABLE public.t
(
  a timestamp,
  b int,
  c varchar(10),
  d int DEFAULT date_part('month', t.a)
);
```

4. Query table `t`:

```
=> select * from t;
      a          | b | c | d
-----+-----+-----+-----
2012-05-14 10:39:25 | 2 | MA | 5
(1 row)
```

Column d returns integer 5 (month of May).

Default column value derived from user-defined scalar function

This example shows a user-defined scalar function that adds two integer values. The function is called `add2ints` and takes two arguments.

1. Develop and deploy the function, as described in [Scalar Functions \(UDSFs\)](#).
2. Create a sample table, `t1`, with two integer columns:

```
=> CREATE TABLE t1 ( x int, y int );  
CREATE TABLE
```

3. Insert some values into `t1`:

```
=> insert into t1 values (1,2);  
OUTPUT  
-----  
      1  
(1 row)  
=> insert into t1 values (3,4);  
OUTPUT  
-----  
      1  
(1 row)
```

4. Use `ALTER TABLE` to add a column to `t1`, with the default column value derived from the UDSF `add2ints`:

```
alter table t1 add column z int default add2ints(x,y);  
ALTER TABLE
```

5. List the new column:

```
select z from t1;  
z  
----  
  3  
  7  
(2 rows)
```

Table with a `SET USING` column that queries another table for its values

1. Define tables `t1` and `t2`. Column `t2.b` is defined to get its data from column `t1.b`, through the query in its `SET USING` clause:

```
=> CREATE TABLE t1 (a INT PRIMARY KEY ENABLED, b INT);
CREATE TABLE

=> CREATE TABLE t2 (a INT, alpha VARCHAR(10), b INT SET USING (SELECT t1.b FROM t1 WHERE
t1.a=t2.a))
ORDER BY a SEGMENTED BY HASH(a) ALL NODES;
CREATE TABLE
```

Important: The definition for table t2 includes `SEGMENTED BY` and `ORDER BY` clauses that exclude `SET USING` column b. If these clauses are omitted, Vertica creates an [auto-projection](#) for this table that specifies column b in its `SEGMENTED BY` and `ORDER BY` clauses. Inclusion of a `SET USING` column in any projection's segmentation or sort order prevents function `REFRESH_COLUMNS` from populating this column. Instead, it returns with an error.

For details on this and other restrictions, see [REFRESH_COLUMNS](#).

2. Populate the tables with data:

```
=> INSERT INTO t1 VALUES(1,11);
=> INSERT INTO t1 VALUES(2,22);
=> INSERT INTO t1 VALUES(3,33);
=> INSERT INTO t1 VALUES(4,44);
=> INSERT INTO t2 VALUES(1, 'aa');
=> INSERT INTO t2 VALUES(2, 'bb');
=> COMMIT;
COMMIT
```

3. View the data in table t2: Column in `SET USING` column b is empty, pending invocation of Vertica function `REFRESH_COLUMNS`:

```
=> SELECT * FROM t2;
 a | alpha | b
---+-----+---
 1 | aa    |
 2 | bb    |
(2 rows)
```

4. Refresh the column data in table t2 by calling function `REFRESH_COLUMNS`:

```
=> SELECT REFRESH_COLUMNS ('t2','b', 'REBUILD');
REFRESH_COLUMNS
-----
refresh_columns completed
(1 row)
```

In this example, `REFRESH_COLUMNS` is called with the optional argument `REBUILD`. This argument specifies to replace all data in `SET USING` column b. It is generally good practice

to call `REFRESH_COLUMNS` with `REBUILD` on any new `SET USING` column. For details, see [REFRESH_COLUMNS](#).

5. View data in refreshed column `b`, whose data is obtained from table `t1` as specified in the column's `SET USING` query:

```
=> SELECT * FROM t2 ORDER BY a;  
a | alpha | b  
-----  
1 | aa    | 11  
2 | bb    | 22  
(2 rows)
```

Altering Table Definitions

Using `ALTER TABLE` syntax, you can respond to your evolving database schema requirements. The ability to change the definition of existing database objects facilitates ongoing maintenance. Furthermore, most of these options are both fast and efficient for large tables, because they consume fewer resources and less storage than having to stage data in a temporary table.

`ALTER TABLE` lets you perform the following table-level changes:

- [Add](#) and [drop](#) table columns.
- [Rename a table](#).
- [Add](#) and [drop](#) constraints.
- [Alter key constraint enforcement](#).
- [Move a table to a new schema](#).
- [Change a table owner](#).
- [Change and reorganize table partitions](#).

`ALTER TABLE` has an `ALTER COLUMN` clause that lets you modify column definitions—for example, change their name or data type. For column-level changes, see [Managing Table Columns](#).

Exclusive ALTER TABLE Clauses

The following ALTER TABLE clauses are exclusive: you cannot combine them with another ALTER TABLE clause:

- ADD COLUMN
- RENAME COLUMN
- SET SCHEMA
- PARTITION BY
- REORGANIZE
- REMOVE PARTITIONING
- RENAME [TO]
- OWNER TO

Note: You can use the ADD constraints and DROP constraints clauses together.

Using Consecutive ALTER TABLE Commands

With the exception of performing a table rename, complete ALTER TABLE statements consecutively. For example, to add multiple columns to a table, issue consecutive ALTER TABLE ADD COLUMN statements.

External Table Restrictions

Not all ALTER TABLE options pertain to external tables. For instance, you cannot add a column to an external table, but you can rename the table:

```
=> ALTER TABLE mytable RENAME TO mytable2;  
ALTER TABLE
```

Restoring to an Earlier Epoch

If you restore the database to an epoch that precedes changes to the table definition, the restore operation reverts the table to the earlier definition. For example, if you change a column's data type from CHAR(8) to CHAR(16) in epoch 10, and then restore the database from epoch 5, the column reverts to CHAR(8) data type.

Adding Table Columns

You add a column to a persistent table with the `ALTER TABLE` clause `ADD COLUMN`:

```
ALTER TABLE
...
ADD COLUMN column-name datatype
    [column-constraint]
    [ENCODING encoding-type]
    [RESTRICT | CASCADE]
    [PROJECTIONS (projection-name [,...]) ]
```

Note: Before you can add columns to a table, all superprojections that are anchored to it must be up to date.

Default Add Operations

When you update a table with a new columns, Vertica always executes the following actions:

- Refreshes existing rows with their default values. For example, if you set a column's default expression to `CURRENT_TIMESTAMP`, Vertica updates all rows with the current timestamp.
- Adds the column to all superprojections that are anchored to the updated table.

Table Locking

When you use `ADD COLUMN` to alter a table, Vertica takes an O lock on the table until the operation completes. The lock prevents `DELETE`, `UPDATE`, `INSERT`, and `COPY` statements from accessing the table. The lock also blocks `SELECT` statements issued at `SERIALIZABLE` isolation level, until the operation completes.

If you use `CASCADE`, Vertica also takes O locks on all anchor tables of any pre-join projections associated with the target table. Consequently, `SELECT` statements issued on those tables at `SERIALIZABLE` isolation level are blocked until the operation completes.

Adding a column to a table does not affect K-safety of the physical schema design.
You can add columns when nodes are down.

Using the PROJECTIONS Option to Add New Columns to Projections

When you add a new column to a table, you can use the PROJECTIONS option to simultaneously add the new column to one or more existing projections.

Note: The PROJECTIONS option accepts the base name of a projection. When you specify the base name, Vertica updates the projection along with any buddy projections.

The following example creates a table and projection, and adds a new column to the table and to the projection.

1. Create a sample table and add content.

```
=> CREATE TABLE t1 (col1 int, col2 int, col3 int);  
=> INSERT INTO t1 VALUES (1, 2, 3);  
=> INSERT INTO t1 VALUES (4, 5, 6);
```

2. Create a projection.

```
=> CREATE PROJECTION myproj (col1) AS SELECT col1 FROM t1;
```

3. Add a new column to the table, simultaneously adding the column to the existing projection.

```
=> ALTER TABLE t1 ADD COLUMN newcol INT PROJECTIONS(myproj);
```

4. Check that the new column is included in the projection.

```
=> SELECT table_name, table_column_name, projection_name FROM projection_columns WHERE  
projection_name LIKE 'myproj%';
```

table_name	table_column_name	projection_name
t1	col1	myproj_b0
t1	col1	myproj_b1
t1	newcol	myproj_b0
t1	newcol	myproj_b1

(4 rows)

The following example adds a new column and defines the column with a column-constraint DEFAULT expression. (For details about expressions, see [Defining Column Values](#).) When you include a DEFAULT expression, the columns that you reference in the expression must have already been included in the existing projection. If the columns from the table you are altering are referenced in the expression, but were not included in the existing projection, Vertica rolls back. The following example illustrates.

1. Create two tables and add content.

```
=> CREATE TABLE dimension (pk INT, val INT, colx INT);
=> CREATE TABLE fact (pk INT, fk INT, cols INT);
=> INSERT INTO dimension VALUES (1, 2, 3);
=> INSERT INTO fact VALUES (4, 5, 6);
```

2. Create a projection that includes the pk column from the table fact.

```
=> CREATE PROJECTION fact_pk AS SELECT pk FROM fact;
```

3. Attempt to add a new column to the table fact. Note that the sample references the column fact.fk, which is not included in the existing projection. Vertica rolls back.

```
=> ALTER TABLE fact ADD COLUMN val INT
  DEFAULT (SELECT val FROM dimension WHERE dimension.pk = fact.fk) PROJECTIONS (fact_pk);
ROLLBACK 7871: Projection "public.fact_pk_b0" refers to column "fk" referenced in the added
column's default expression
```

What follows is the same example, but in this case the projection includes both columns.

1. Create a sample table and add content.

```
=> CREATE TABLE t1 (col1 int, col2 int, col3 int);
=> INSERT INTO t1 VALUES (1, 2, 3);
=> INSERT INTO t1 VALUES (4, 5, 6);
```

2. Create a projection that includes both the pk and fk columns from the table fact.

```
=> CREATE PROJECTION fact_pk AS SELECT pk, fk FROM fact;
```

3. Add the new column using a column-constraint DEFAULT expression.

```
=> ALTER TABLE fact ADD COLUMN val INT
  DEFAULT (SELECT val FROM dimension WHERE dimension.pk = fact.fk) PROJECTIONS (fact_pk);
```

4. Check that the new column val is included in the projection.

```
=> SELECT table_name, table_column_name, projection_name FROM projection_columns WHERE  
projection_name LIKE 'fact_pk%';
```

table_name	table_column_name	projection_name
fact	pk	fact_pk_b0
fact	pk	fact_pk_b1
fact	fk	fact_pk_b0
fact	fk	fact_pk_b1
fact	val	fact_pk_b0
fact	val	fact_pk_b1

Updating Associated Table Views

Adding new columns to a table that has an associated view does not update the view's result set, even if the view uses a wildcard (*) to represent all table columns. To incorporate new columns, you must [recreate the view](#).

Dropping Table Columns

When an ALTER TABLE statement includes a DROP COLUMN clause to drop a column, Vertica drops the specified column from the table and the ROS containers that correspond to the dropped column.

The syntax looks like this:

```
ALTER TABLE [schema.]table-name DROP [ COLUMN ] column-name [ CASCADE | RESTRICT ]
```

After a DROP COLUMN operation completes, data backed up from the current epoch onward recovers without the column. Data recovered from a backup that precedes the current epoch re-add the table column. Because drop operations physically purge object storage and catalog definitions (table history) from the table, AT EPOCH (historical) queries return nothing for the dropped column.

The altered table retains its object ID.

Note: Drop column operations can be fast because these catalog-level changes do not require data reorganization, so Vertica can quickly reclaim disk storage.

Restrictions

- You cannot drop or alter a primary key column or a column that participates in the table partitioning clause.

- You cannot drop the first column of any projection sort order, or columns that participate in a projection segmentation expression.
- All nodes must be up.
- You cannot drop a column associated with an access policy. Attempts to do so produce the following error:
ERROR 6482: Failed to parse Access Policies for table "t1"

Using CASCADE to Force a Drop

If the table column to drop has dependencies, you must qualify the `DROP COLUMN` clause with the `CASCADE` option. For example, the target column might be specified in a projection sort order. In this and other cases, `DROP COLUMN . . CASCADE` handles the dependency by reorganizing catalog definitions or dropping a projection. In all cases, `CASCADE` performs the minimal reorganization required to drop the column.

Use `CASCADE` to drop a column with the following dependencies:

Dropped column dependency	CASCADE behavior
Any constraint	Vertica drops the column when a FOREIGN KEY constraint depends on a UNIQUE or PRIMARY KEY constraint on the referenced columns.
Specified in projection sort order	Vertica truncates projection sort order up to and including the projection that is dropped without impact on physical storage for other columns and then drops the specified column. For example if a projection's columns are in sort order (a,b,c), dropping column b causes the projection's sort order to be just (a), omitting column (c).
Specified in one of the following: <ul style="list-style-type: none"> • Pre-join projection • Projection segmentation expression 	<p>In both cases, the column to drop is integral to the projection definition. If possible, Vertica drops the projections as long as doing so does not compromise K-safety; otherwise, the transaction rolls back.</p> <p>For example, a table has multiple projections, where one projection's segmentation clause specifies the target column. Vertica tries to drop this projection, unless doing so violates K-safety. In this case, the transaction rolls back.</p>

Dropped column dependency	CASCADE behavior
Referenced as default value of another column	See Dropping a Column Referenced as Default , below.

Dropping a Column Referenced as Default

You might want to drop a table column that is referenced by another column as its default value. For example, the following table is defined with two columns, a and b;, where b gets its default value from column a:

```
=> CREATE TABLE x (a int) UNSEGMENTED ALL NODES;  
CREATE TABLE  
=> ALTER TABLE x ADD COLUMN b int DEFAULT a;  
ALTER TABLE
```

In this case, dropping column a requires the following procedure:

1. Remove the default dependency through ALTER COLUMN. .DROP DEFAULT:

```
=> ALTER TABLE x ALTER COLUMN b DROP DEFAULT;
```

2. Create a replacement superprojection for the target table if one or both of the following conditions is true:
 - The target column is the table's first sort order column. If the table has no explicit sort order, the default table sort order specifies the first table column as the first sort order column. In this case, the new superprojection must specify a sort order that excludes the target column.
 - If the table is segmented, the target column is specified in the segmentation expression. In this case, the new superprojection must specify a segmentation expression that excludes the target column.

Given the previous example, table x has a default sort order of (a,b). Because column a is the table's first sort order column, you must create a replacement superprojection that is sorted on column b:

```
=> CREATE PROJECTION x_p1 as select * FROM x ORDER BY b UNSEGMENTED ALL NODES;
```

3. Run `START_REFRESH`:

```
=> SELECT START_REFRESH();
          START_REFRESH
-----
Starting refresh background process.

(1 row)
```

4. Run `MAKE_AHM_NOW`:

```
=> SELECT MAKE_AHM_NOW();
          MAKE_AHM_NOW
-----
AHM set (New AHM Epoch: 1231)

(1 row)
```

5. Drop the column:

```
=> ALTER TABLE x DROP COLUMN a CASCADE;
```

Vertica implements the `CASCADE` directive as follows:

- Drops the original superprojection for table `x` (`x_super`).
- Updates the replacement superprojection `x_p1` by dropping column `a`.

Examples

The following series of commands successfully drops a `BYTEA` data type column:

```
=> CREATE TABLE t (x BYTEA(65000), y BYTEA, z BYTEA(1));
CREATE TABLE
=> ALTER TABLE t DROP COLUMN y;
ALTER TABLE
=> SELECT y FROM t;
ERROR 2624: Column "y" does not exist
=> ALTER TABLE t DROP COLUMN x RESTRICT;
ALTER TABLE
=> SELECT x FROM t;
ERROR 2624: Column "x" does not exist
=> SELECT * FROM t;
  z
---
(0 rows)
=> DROP TABLE t CASCADE;
DROP TABLE
```

The following series of commands tries to drop a FLOAT(8) column and fails because there are not enough projections to maintain K-safety.

```
=> CREATE TABLE t (x FLOAT(8),y FLOAT(08));  
CREATE TABLE  
=> ALTER TABLE t DROP COLUMN y RESTRICT;  
ALTER TABLE  
=> SELECT y FROM t;  
ERROR 2624: Column "y" does not exist  
=> ALTER TABLE t DROP x CASCADE;  
ROLLBACK 2409: Cannot drop any more columns in t  
=> DROP TABLE t CASCADE;
```

Altering Constraint Enforcement

To alter how Vertica enforces constraints, use the [ALTER TABLE](#) clause ALTER CONSTRAINT. You must qualify this clause with the keyword ENABLED or DISABLED:

- ENABLED automatically enforces a primary key, unique key or check constraint.
- DISABLED prevents automatic enforcement of a primary key, unique key or check constraint.

For more information on automatic constraint enforcement, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#)

If you disable automatic enforcement of primary or unique key constraints, you can instead run [ANALYZE_CONSTRAINTS](#) to verify that columns have unique values after running a DML command or bulk loading. In the case of check constraints, you can use ANALYZE_CONSTRAINTS to validate check constraint conditions.

See [About Constraints](#) for general information about constraints.

Renaming Tables

The ALTER TABLE RENAME TO statement lets you rename one or more tables. Renaming tables does not change the table OID.

You rename multiple tables by supplying two comma-delimited lists. Vertica maps the names according to their order in the two lists. Only the first list can qualify table names with a schema. For example:

```
=> ALTER TABLE S1.T1, S1.T2 RENAME TO U1, U2;
```

The `RENAME TO` parameter is applied atomically : all tables are renamed, or none of them. For example, if the number of tables to rename does not match the number of new names, none of the tables is renamed.

Note: Renaming a table referenced by a view causes the view to fail, unless you create another table with the previous name to replace the renamed table.

Using Rename to Swap Tables Within a Schema

You can use `ALTER TABLE RENAME TO` to swap tables within the same schema, without actually moving data. You cannot swap tables across schemas.

To swap tables within a schema (example statement is split to explain steps):

1. Enter the names of the tables to swap, followed by a new temporary table placeholder (temps):

```
=> ALTER TABLE T1, T2, temps
```

2. Use the `RENAME TO` clause to swap the tables: T1 to temps, T2 to T1, and temps to T2:

```
RENAME TO temps, T1, T2;
```

Moving Tables to Another Schema

The `ALTER TABLE` clause `SET SCHEMA` moves a table from one schema to another. Vertica automatically moves all projections that are anchored to the source table to the destination schema.

Moving a table across schemas requires that you have `USAGE` privileges on the current schema and `CREATE` privileges on destination schema. You can move only one table between schemas at a time. You cannot move temporary tables across schemas.

Name Conflicts

If a table of the same name or any of the projections that you want to move already exist in the new schema, the statement rolls back and does not move either the table or any projections.

To work around name conflicts:

1. Rename any conflicting table or projections that you want to move.
2. Run the `ALTER TABLE SET SCHEMA` statement again.

Note: Vertica lets you move system tables to system schemas. Moving system tables could be necessary to support designs created through the Database Designer.

Example

The following example moves table T1 from schema S1 to schema S2. All projections that are anchored on table T1 automatically move to schema S2:

```
=> ALTER TABLE S1.T1 SET SCHEMA S2;
```

Changing Table Ownership

As a superuser or table owner, you can reassign table ownership with `ALTER TABLE`, as follows:

```
ALTER TABLE [schema.]table-name OWNER TO owner-name
```

Changing table ownership is useful when moving a table from one schema to another. Ownership reassignment is also useful when a table owner leaves the company or changes job responsibilities. Because you can change the table owner, the tables won't have to be completely rewritten, you can avoid loss in productivity.

Changing table ownership automatically causes the following changes:

- Grants on the table that were made by the original owner are dropped and all existing privileges on the table are revoked from the previous owner. Changes in table ownership has no effect on schema privileges.
- Ownership of dependent `IDENTITY/AUTO-INCREMENT` sequences are transferred with the table. However, ownership does not change for independent sequences created with `CREATE SEQUENCE`. To transfer ownership of these sequences, use `ALTER SEQUENCE`.
- New table ownership is propagated to its projections.

Example

In this example, user Bob connects to the database, looks up the tables, and transfers ownership of table t33 from himself to user Alice.

```
=> \c - Bob
You are now connected as user "Bob".
=> \d
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
 public | t33   | table | Bob    |
(2 rows)
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
```

Notice that when Bob looks up database tables again, he no longer sees table t33.

```
=> \d
          List of tables
          List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
(1 row)
```

When user Alice connects to the database and looks up tables, she sees she is the owner of table t33.

```
=> \c - Alice
You are now connected as user "Alice".
=> \d
          List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | t33  | table | Alice |
(2 rows)
```

Either Alice or a superuser can transfer table ownership back to Bob. In the following case a superuser performs the transfer.

```
=> \c - dbadmin
You are now connected as user "dbadmin".
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> \d
          List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
 public | comments | table | dbadmin |
 public | t33   | table | Bob    |
 s1    | t1    | table | User1  |
```

(4 rows)

You can also query system table [V_CATALOG.TABLES](#) to view table and owner information. Note that a change in ownership does not change the table ID.

In the below series of commands, the superuser changes table ownership back to Alice and queries the TABLES system table.

```
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM tables;
table_schema_id | table_schema | table_id | table_name | owner_id | owner_name
-----+-----+-----+-----+-----+-----
45035996273704968 | public | 45035996273713634 | applog | 45035996273704962 | dbadmin
45035996273704968 | public | 45035996273724496 | comments | 45035996273704962 | dbadmin
45035996273730528 | s1 | 45035996273730548 | t1 | 45035996273730516 | User1
45035996273704968 | public | 45035996273795846 | t33 | 45035996273724576 | Alice
(5 rows)
```

Now the superuser changes table ownership back to Bob and queries the TABLES table again. Nothing changes but the owner_name row, from Alice to Bob.

```
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM tables;
table_schema_id | table_schema | table_id | table_name | owner_id | owner_name--
-----+-----+-----+-----+-----+-----
45035996273704968 | public | 45035996273713634 | applog | 45035996273704962 | dbadmin
45035996273704968 | public | 45035996273724496 | comments | 45035996273704962 | dbadmin
45035996273730528 | s1 | 45035996273730548 | t1 | 45035996273730516 | User1
45035996273704968 | public | 45035996273793876 | foo | 45035996273724576 | Alice
45035996273704968 | public | 45035996273795846 | t33 | 45035996273714428 | Bob
(5 rows)
```

See Also

[Changing Sequence Ownership](#)

Working with Sequence Types

Vertica supports these types of incrementing values:

- **Auto-increment or Identity column:** The Auto-increment and Identity types are identical. They are basic incrementing numeric column types that the database increments automatically.
- **Named sequences:** A named sequence is a database object that generates unique numbers in ascending or descending sequential order.

You define Auto-increment and Identity sequences through [column constraints](#) in the [CREATE TABLE](#) statement. These sequences are incremented each time a row is added to the table. Both of these sequence object types are table-dependent and do not persist independently. Vertica does not roll back an identity value even if a transaction that tries to insert a value is not committed. The [LAST_INSERT_ID](#) function returns the last value generated for an auto-increment or identity column.

Each type of sequence value has a set of properties. A named sequence has the most properties, and an Auto-increment sequence the least. The following table lists the differences between the three sequence values:

Behavior	Named Sequence	Identity	Auto-increment
Default cache value 250K	X	X	X
Set initial cache	X	X	X
Define start value	X	X	X
Specify increment unit	X	X	X
Exists as an independent object	X		
Exists only as part of table		X	X
Create as column constraint		X	X
Requires name	X		
Use in expressions	X		
Unique across tables	X		

Behavior	Named Sequence	Identity	Auto-increment
Change parameters	X		
Move to different schema	X		
Set to increment or decrement	X		
Grant privileges to object	X		
Specify minimum value	X		
Specify maximum value	X		

Note: While sequence object values are guaranteed to be unique, they are not guaranteed to be contiguous. Since sequences are not necessarily contiguous, you may interpret the returned values as missing. For example, two nodes can increment a sequence at different rates. The node with a heavier processing load increments the sequence, but the values are not contiguous with those being incremented on a node with less processing.

Using Named Sequences

You use named sequences most often when an application requires a unique identifier in a table or an expression. Once a named sequence returns a value, it never returns that same value again in the same session. Named sequences are independent objects. While you can use their values in tables, they are not subordinate to the tables in which you use the named sequences.

This section includes:

- [Creating and Instantiating Named Sequences](#)
- [Distributing Named Sequences](#)
- [Altering Sequences](#)
- [Changing Sequence Ownership](#)
- [Dropping Sequences](#)

Creating and Instantiating Named Sequences

You create a named sequence with `CREATE SEQUENCE`. The statement requires only a sequence name; all other parameters are optional.

The following example creates an ascending named sequence, `my_seq`, starting at the value 100:

```
=> create sequence my_seq START 100;  
CREATE SEQUENCE
```

After creating a sequence, call the function `NEXTVAL` at least once in a session to create a cache for the sequence and its initial value. Subsequently, use `NEXTVAL` to increment the sequence value. Call the function `CURRVAL` to get the current value.

The following call to `NEXTVAL` instantiates the newly-created `my_seq` sequence and sets its first number:

```
=> SELECT NEXTVAL('my_seq');  
nextval  
-----  
      100  
(1 row)
```

If you call `CURRVAL` before `NEXTVAL`, the system returns an error:

```
dbt=> SELECT CURRVAL('my_seq');  
ERROR 4700: Sequence my_seq has not been accessed in the session
```

You can use a sequence as part of creating a table. The sequence must already exist, and have been instantiated using the `NEXTVAL` statement.

Update sequence number values by calling the `NEXTVAL` function, which increments/decrements the current sequence and returns the next value. Use `CURRVAL` to return the current value. These functions can also be used in `INSERT` and `COPY` expressions.

The following example shows creation of a simple table including a named sequence. Table creation is followed by an insert into the table using the sequence as the default value for the column `ID`:

```
CREATE TABLE customer2(  
  ID INTEGER DEFAULT NEXTVAL('my_seq'),  
  lname VARCHAR(25),  
  fname VARCHAR(25),  
  membership_card INTEGER  
);  
INSERT INTO customer2 VALUES (default, 'Carr', 'Mary', 87432);
```

Now query the table you just created. The column named ID has been incremented by (1) again to 104:

```
SELECT * FROM customer2;
 ID | lname | fname | membership_card
-----+-----+-----+-----
 104 | Carr  | Mary  |                87432
(1 row)
```

Using Named Sequence Functions

When you create a named sequence object, you can also specify its increment or decrement value. The default is 1.

Use these functions with named sequences:

- **NEXTVAL** advances the sequence and returns the next value. The value is incremented for ascending sequences and decremented for descending. The first time you call NEXTVAL after creating a named sequence, the function sets up the default or specified amount of cache on each cluster node. From its cache store, each node returns either the default sequence value, or a start number you specified with CREATE SEQUENCE.
- **CURRVAL** returns the last value that the previous invocation of NEXTVAL returned in the current session. If there were no calls to NEXTVAL after creating a sequence, the CURRVAL function returns an error:

```
dbt=> create sequence seq2;
CREATE SEQUENCE
dbt=> select currval('seq2');
ERROR 4700: Sequence seq2 has not been accessed in the session
```

Using DDL Statements with Named Sequences

The table that follows lists the statements you use specifically for named sequences.

Use this statement...	To...
CREATE SEQUENCE	Create a named sequence object.
ALTER SEQUENCE	Alter named sequence parameters, rename a sequence within a schema, or move a named sequence between schemas.
DROP SEQUENCE	Remove a named sequence object.
GRANT SEQUENCE	Grant user privileges to a named sequence object. See also Sequence Privileges .

Distributing Named Sequences

When you create a named sequence, the CACHE parameter determines the number of sequence values each node maintains during a session. The default cache value is 250K, so each node reserves 250,000 values per session for each sequence. The default cache size provides an efficient means for large insert or copy operations. Specifying a smaller number of cache values can impact performance of large loads, since Vertica must create a new set of cache values whenever more are required. Getting more cache for a new set of sequence values requires Vertica to perform a catalog lock. Such locks can adversely affect database performance, since some activities, such as data inserts, cannot occur until Vertica releases the lock.

By default, when the cache is initially empty, the initiator node requests and reserves cache for all nodes in a cluster. You can change this default so that each node requests its own cache. To change the default, set the parameter ClusterSequenceCacheMode to 0. For information on how Vertica requests and distributes cache among all nodes in a cluster, refer to [How Vertica Allots Cache for Sequencing](#).

Effects of Distributed Sessions

Vertica distributes a session across all nodes. After you create a named sequence, the first time any cluster node executes a NEXTVAL () statement within a query, the node requests its own cache of sequence values. The node then maintains that set of values for the current session. Other nodes executing a NEXTVAL () statement create and maintain their own cache of sequence values.

During a session, nodes can increment sequence values from NEXTVAL () statements at different rates. This behavior results in the sequences from a NEXTVAL statement on one node not being sequential with sequence values from another node. Each sequence is guaranteed to be unique, but can be out of order with a NEXTVAL statement executed on another node. Regardless of the number of calls to NEXTVAL and CURRVAL, Vertica increments a sequence only once per row. If multiple calls to NEXTVAL occur in the same row, the statement returns the same value.

If sequences are used in join statements, Vertica increments a sequence once for each composite row output by the join.

Calculating Named Sequences

Vertica calculates the current value of a sequence as follows:

- At the end of every statement, the state of all sequences used in the session is returned to the initiator node.
- The initiator node calculates the maximum CURRVAL of each sequence across all states on all nodes.
- This maximum value is used as CURRVAL in subsequent statements until another NEXTVAL is invoked.

Losing Sequence Values

Sequence values in cache can be lost in the following situations:

- If a statement fails after NEXTVAL is called (thereby consuming a sequence value from the cache), the value is lost.
- If a disconnect occurs (for example, dropped session), any remaining values in cache that have not been returned through NEXTVAL are lost.
- When the initiator node distributes a new block of cache to each node where one or more nodes has not used up its current cache allotment. For information on this scenario, refer to [How Vertica Allots Cache for Sequencing](#).

To recover lost sequence values, you can run an [ALTER SEQUENCE](#) command to define a new sequence number generator, which resets the counter to the correct value in the next session.

See Also

- [Sequence Privileges](#)
- [ALTER SEQUENCE](#)
- [CREATE TABLE](#)
- [Column-Constraint](#)
- [CURRVAL](#)
- [DROP SEQUENCE](#)
- [GRANT \(Sequence\)](#)
- [NEXTVAL](#)

Altering Sequences

The **ALTER SEQUENCE** statement lets you change the attributes of a previously-defined named sequence. Changes take effect in the next database session. Any parameters not specifically set in the ALTER SEQUENCE command retain their previous settings.

The ALTER SEQUENCE statement lets you rename an existing sequence, or the schema of a sequence, but you cannot combine either of these changes with any other optional parameters.

Note: Using ALTER SEQUENCE to set a START value below the CURRVAL can result in duplicate keys.

Examples

The following example modifies an ascending sequence called `my_seq` to start at 105:

```
=>ALTER SEQUENCE my_seq RESTART WITH 105;
```

The following example moves a sequence from one schema to another:

```
=>ALTER SEQUENCE [public.]my_seq SET SCHEMA vmart;
```

The following example renames a sequence in the Vmart schema:

```
=>ALTER SEQUENCE [vmart.]my_seq RENAME TO serial;
```

Remember that changes occur only after you start a new database session. For example, if you create a named sequence `my_sequence`, starting at value 10, each time you call `NEXTVAL()`, you increment the value by 1, as in the following series of commands:

```
=>CREATE SEQUENCE my_sequence START 10;
=>SELECT NEXTVAL('my_sequence');
  nextval
-----
      10
(1 row)
=>SELECT NEXTVAL('my_sequence');
  nextval
-----
      11
(1 row)
```

Next, issue the ALTER SEQUENCE statement to assign a new value starting at 50:

```
=>ALTER SEQUENCE my_sequence RESTART WITH 50;
```

When you call the NEXTVAL function, the sequence is incremented again by 1 value:

```
=>SELECT NEXTVAL('my_sequence');
NEXTVAL
-----
      12
(1 row)
```

The sequence starts at 50 only after restarting the database session:

```
=>SELECT NEXTVAL('my_sequence');
NEXTVAL
-----
      50
(1 row)
```

Changing Sequence Ownership

The ALTER SEQUENCE command lets you change the attributes of an existing sequence. All changes take effect immediately, within the same session. Any parameters not set during an ALTER SEQUENCE statement retain their prior settings.

If you need to change sequence ownership, such as if an employee who owns a sequence leaves the company, you can do so with the following ALTER SEQUENCE syntax:

```
=> ALTER SEQUENCE sequence-name OWNER TO new-owner-name;
```

This operation immediately reassigns the sequence from the current owner to the specified new owner.

Only the sequence owner or a superuser can change ownership, and reassignment does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Note: Renaming a table owner transfers ownership of dependent sequence objects (associated IDENTITY/AUTO-INCREMENT sequences) but does not transfer ownership of other referenced sequences. See [Changing Table Ownership](#).

Example

The following example reassigns sequence ownership from the current owner to user Bob:

```
=> ALTER SEQUENCE sequential OWNER TO Bob;
```

See [ALTER SEQUENCE](#) in the SQL Reference Manual for details.

Dropping Sequences

Use `DROP SEQUENCE` to remove a named sequence. You cannot drop a sequence in if any of the following conditions is true:

- Other objects depend on the sequence. `DROP SEQUENCE` does not support cascade operations.
- A column's `DEFAULT` expression references the sequence. Before dropping the sequence, you must remove all column references to it.

Example

The following command drops the sequence named `my_sequence`:

```
=> DROP SEQUENCE my_sequence;
```

Using `AUTO_INCREMENT` and `IDENTITY` Sequences

You can define a column that automatically increments its value as new rows are added, with options `AUTO_INCREMENT` or `IDENTITY`. See `CREATE TABLE` [Column-Constraint](#) for syntax options.

While sequence object values are guaranteed to be unique, they are not guaranteed to be contiguous. Since sequences are not necessarily contiguous, you may interpret the returned values as missing. For example, two nodes can increment a sequence at different rates. The node with a heavier processing load increments the sequence, but the values are not contiguous with those being incremented on a node with less processing.

`IDENTITY` and `AUTO_INCREMENT` can take between 0 and three arguments, which Vertica parses as follows:

# arguments	Description
None	Specifies starting column values at 1. Values increment by 1, but values are not guaranteed to be contiguous. Your number of nodes and cache impact sequence values. Setting cache to 1 (no cache) can ensure contiguous values.
1	Specifies cache size, an integer value ≥ 1 that determines how many unique numbers each node is allocated per session:

	<p>IDENTITY(<i>cache</i>)</p> <p>Default: 250,000</p>
2 or 3	<p>Set as follows:</p> <p>IDENTITY (<i>start</i>, <i>increment</i>[, <i>cache</i>])</p> <ul style="list-style-type: none"> • <i>start</i>: The first value of the IDENTITY column <p>Default: 1</p> <ul style="list-style-type: none"> • <i>increment</i>: How much to increment the value from the previous row's value. <p>Default: 1</p> <ul style="list-style-type: none"> • <i>cache</i>: Integer value ≥ 1 that specifies how many unique numbers each node caches per session <ul style="list-style-type: none"> • Default: 250,000

Note: The *increment* value is the minimum increment. For example, setting an increment value of 3 guarantees that values returned increment by at least 3. However, the values returned may not be contiguous, unless you have set the cache value to 1 (no cache).

The following restrictions apply to IDENTITY and AUTO_INCREMENT sequences:

- You can set this constraint on only one table column.
- IDENTITY and AUTO_INCREMENT values are never rolled back, even if a transaction that tries to insert a value into a table is not committed.
- You cannot change the value of an IDENTITY or AUTO_INCREMENT column in an existing table.

Examples

The following example shows how you can use the IDENTITY column-constraint to create a table with an ID column. The ID column has an initial value of 1. It is incremented by 1 every time a row is inserted.

```
=> CREATE TABLE Premium_Customer(
  ID IDENTITY(1,1),
  lname VARCHAR(25),
```

```
    fname VARCHAR(25),
    store_membership_card INTEGER
  );
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card )
VALUES ('Gupta', 'Saleem', 475987);
```

Confirm the row you added and see the ID value:

```
=> SELECT * FROM Premium_Customer;
 ID | lname | fname | store_membership_card
-----+-----+-----+-----
  1 | Gupta | Saleem |                475987
(1 row)
```

Now add another row:

```
=> INSERT INTO Premium_Customer (lname, fname, store_membership_card)
VALUES ('Lee', 'Chen', 598742);
```

Calling the `LAST_INSERT_ID` function returns value 2 because you previously inserted a new customer (Chen Lee), and this value is incremented each time a row is inserted:

```
=> SELECT LAST_INSERT_ID();
 last_insert_id
-----
                2
(1 row)
```

View all the ID values in the `Premium_Customer` table:

```
=> SELECT * FROM Premium_Customer;
 ID | lname | fname | store_membership_card
-----+-----+-----+-----
  1 | Gupta | Saleem |                475987
  2 | Lee   | Chen   |                598742
(2 rows)
```

The next examples illustrate the three valid ways to use `IDENTITY` arguments. (Note that these examples are valid for the `AUTO_INCREMENT` argument also.) The first example uses a cache of 100, and the defaults for start value (1) and increment value (1):

```
=> CREATE TABLE t1(x IDENTITY(100), y INT);
```

The second example specifies the start and increment values as 1, and defaults to a cache value of 250,000:

```
=> CREATE TABLE t2(y IDENTITY(1,1), x INT);
```

The third example specifies start and increment values of 1, and a cache value of 100:

```
=> CREATE TABLE t3(z IDENTITY(1,1,100), zx INT);
```

Using a Named Sequence with an AUTO_INCREMENT or IDENTITY Column

A table can contain only one AUTO_INCREMENT or IDENTITY column. A table with an AUTO_INCREMENT or IDENTITY column can also contain a named sequence. In the following example, table test2 contains a named sequence (my_seq), and an auto_increment value for the column last):

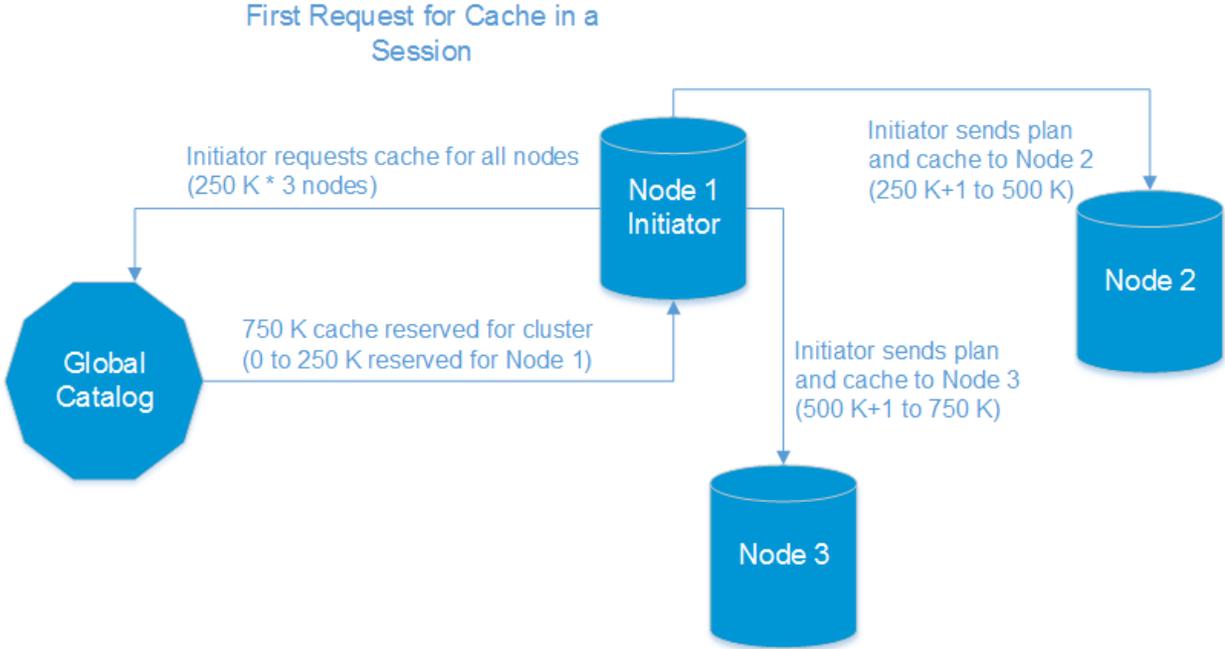
```
VMart=> CREATE TABLE test2 (id INTEGER NOT NULL UNIQUE,  
    middle INTEGER DEFAULT NEXTVAL('my_seq'),  
    next INT, last AUTO_INCREMENT);  
CREATE TABLE
```

How Vertica Allots Cache for Sequencing

Vertica caching is similar for all types of sequences, including named sequences, identity sequences, and auto-increment sequences. To allot cache among the nodes in a cluster using named sequences, Vertica uses the following process.

1. By default, when a session begins, an initiator node requests cache for itself and all other nodes in a cluster.
2. The initiator distributes the cache to other nodes when it distributes the execution plan.
3. Because the initiator requests a block of cache for all nodes, only the initiator locks the global catalog for the cache request.

This approach is optimal for large INSERT-SELECT and COPY operations. The following figure shows the initiator request for cache (with 250-K cache default).



Nodes run out of cache at different times. While executing the same query, nodes individually request additional cache as needed.

For new queries in the same session, the initiator might have an empty cache if it used all of its cache to execute the previous query execution. In this case, the initiator requests cache for all nodes.

You can change how Vertica initially allots cache by setting the configuration parameter `ClusterSequenceCacheMode` to 0 (disabled). When this parameter is set to 0, all nodes in the cluster request their own cache and catalog lock. However, for initial large INSERT-SELECT and COPY operations, when the cache is empty for all nodes, each node requests cache at the same time. These multiple requests result in multiple simultaneous locks on the global catalog, which can adversely affect performance. For this reason, `ClusterSequenceCacheMode` should remain set to its default value of 1 (enabled).

The following example compares how different settings of `ClusterSequenceCacheMode` affect cache allotment for sequence processing. The example assumes a three-node cluster, the default 250 K cache allotment for each node, and sequence ID values that are sequential (incrementing by 1).

Workflow step	<code>ClusterSequenceCacheMode = 1</code>	<code>ClusterSequenceCacheMode = 0</code>
1	Cache is empty for all nodes. Initiator node requests block of 250 K	Cache is empty for all nodes. Each node, including initiator,

Workflow step	ClusterSequenceCacheMode = 1	ClusterSequenceCacheMode = 0
	cache equal to for each node.	requests its own 250 K cache.
2	<p>Each node has cache:</p> <ul style="list-style-type: none"> • Node 1: 0–250 K • Node 2: 250 K + 1 to 500 K • Node 3: 500 K + 1 to 750 K <p>Each node begins to use its cache allotment as it does its work.</p>	
3	<p>Initiator node and node 3 run out of cache.</p> <p>Node 2 only uses 250 K +1 to 400 K, 100 K of cache remains from 400 K +1 to 500 K.</p>	
4	<p>Executing same statement:</p> <ul style="list-style-type: none"> • As individual nodes run out of cache, they individually request a new cache allotment. • If node 2 never uses its cache allotment, the 100-K unused cache becomes a gap in sequence IDs. <p>Executing new statement in same session, if initiator node cache is empty:</p> <ul style="list-style-type: none"> • Initiator node again requests and distributes cache for all nodes. • Nodes receive new cache before their previous allotment is totally used, leaving a gap in the sequence IDs. 	<p>Executing same or new statement:</p> <ul style="list-style-type: none"> • As individual nodes run out of cache, they individually request a new cache allotment. • If node 2 never uses its cache allotment, the 100 K unused cache becomes a gap in sequence IDs.

Merging Table Data

MERGE statements can perform [update](#) and [insert](#) operations on a target table based on the results of a join with a source data set. The join can match a source row with only one target row; otherwise, Vertica returns an error.

MERGE has the following syntax:

```
MERGE INTO target-table USING source-dataset ON join-condition  
      matching-clause[ matching-clause ]
```

Merge operations have at least three components:

- The target table on which to perform update and insert operations. MERGE takes an X (exclusive) lock on the target table until the merge operation is complete.
- [Join to another data set](#), one of the following: a table, view, or subquery result set.
- One or both matching clauses: **WHEN MATCHED THEN UPDATE SET** and **WHEN NOT MATCHED THEN INSERT**.

Basic MERGE Example

In this example, a merge operation involves two tables:

- `visits_daily` logs daily restaurant traffic, and is updated with each customer visit. Data in this table is refreshed every 24 hours.
- `visits_history` stores the history of customer visits to various restaurants, accumulated over an indefinite time span.

Each night, you merge the daily visit count from `visits_daily` into `visits_history`. The merge operation modifies the target table in two ways:

- Updates existing customer data.
- Inserts new rows of data for first-time customers.

One MERGE statement executes both operations as a single (upsert) transaction.

Source and Target Tables

The source and target tables `visits_daily` and `visits_history` are defined as follows:

```
CREATE TABLE public.visits_daily
(
  customer_id int,
  location_name varchar(20),
  visit_time time(0) DEFAULT (now())::timetz(6)
);

CREATE TABLE public.visits_history
(
  customer_id int,
  location_name varchar(20),
  visit_count int
);
```

Table `visits_history` contains rows of three customers who between them visited two restaurants, Etoile and LaRosa:

```
=> SELECT * FROM visits_history ORDER BY customer_id, location_name;
 customer_id | location_name | visit_count
-----+-----+-----
          1001 | Etoile       |           2
          1002 | La Rosa     |           4
          1004 | Etoile       |           1
(3 rows)
```

By close of business, table `visits_daily` contains three rows of restaurant visits:

```
=> SELECT * FROM visits_daily ORDER BY customer_id, location_name;
 customer_id | location_name | visit_time
-----+-----+-----
          1001 | Etoile       | 18:19:29
          1003 | Lux Cafe     | 08:07:00
          1004 | La Rosa     | 11:49:20
(3 rows)
```

Table Data Merge

The following MERGE statement merges `visits_daily` data into `visits_history`:

- For matching customers, MERGE updates the occurrence count.
- For non-matching customers, MERGE inserts new rows.

```
=> MERGE INTO visits_history h USING visits_daily d
  ON (h.customer_id=d.customer_id AND h.location_name=d.location_name)
  WHEN MATCHED THEN UPDATE SET visit_count = h.visit_count + 1
  WHEN NOT MATCHED THEN INSERT (customer_id, location_name, visit_count)
  VALUES (d.customer_id, d.location_name, 1);
OUTPUT
-----
      3
(1 row)
```

MERGE returns the number of rows updated and inserted. In this case, the returned value specifies three updates and inserts:

- Customer 1001's third visit to Etoile
- New customer 1003's first visit to new restaurant Lux Cafe
- Customer 1004's first visit to La Rosa

If you now query table `visits_history`, the result set shows the merged (updated and inserted) data. Updated and new rows are highlighted:

```
=> SELECT * FROM visits_history ORDER BY customer_id, location_name
  customer_id | location_name | visit_count
-----+-----+-----
  1001 | Etoile | 3
  1002 | La Rosa | 4
  1003 | Lux Cafe | 1
  1004 | Etoile | 1
  1004 | La Rosa | 1
(5 rows)
```

MERGE Source Options

A MERGE operation joins the target table to one of the following data sources:

- Another table
- View
- Subquery result set

Merging from Table and View Data

You merge data from one table into another as follows:

```
MERGE INTO target-table USING { source-table | source-view } join-condition  
  matching-clause[ matching-clause ]
```

If you specify a view, Vertica expands the view name to the query that it encapsulates, and uses the result set as the merge source data.

For example, the VMart table `public.product_dimension` contains current and discontinued products. You can move all discontinued products into a separate table `public.product_dimension_discontinued`, as follows:

```
=> CREATE TABLE public.product_dimension_discontinued (  
  product_key int,  
  product_version int,  
  sku_number char(32),  
  category_description char(32),  
  product_description varchar(128));  
  
=> MERGE INTO product_dimension_discontinued tgt  
  USING product_dimension src ON tgt.product_key = src.product_key  
    AND tgt.product_version = src.product_version  
  WHEN NOT MATCHED AND src.discontinued_flag='1' THEN INSERT VALUES  
    (src.product_key,  
     src.product_version,  
     src.sku_number,  
     src.category_description,  
     src.product_description);  
  
OUTPUT  
-----  
  1186  
(1 row)
```

Source table `product_dimension` uses two columns, `product_key` and `product_version`, to identify unique products. The `MERGE` statement joins the source and target tables on these columns in order to return single instances of non-matching rows. The `WHEN NOT MATCHED` clause includes a [filter](#) (`src.discontinued_flag='1'`), which reduces the result set to include only discontinued products. The remaining rows are inserted into target table `product_dimension_discontinued`.

Merging from a Subquery Result Set

You can merge into a table the result set that is returned by a subquery, as follows:

```
MERGE INTO target-table USING (subquery) sq-alias join-condition  
  matching-clause[ matching-clause ]
```

For example, the VMart table `public.product_dimension` is defined as follows (DDL truncated):

```
CREATE TABLE public.product_dimension  
(  
  product_key int NOT NULL,  
  product_version int NOT NULL,
```

```
product_description varchar(128),
sku_number char(32),
...
)
ALTER TABLE public.product_dimension
ADD CONSTRAINT C_PRIMARY PRIMARY KEY (product_key, product_version) DISABLED;
```

Columns `product_key` and `product_version` comprise the table's primary key. You can modify this table so it contains a single column that concatenates the values of these two columns. This column can be used to uniquely identify each product, while also maintaining the original values from `product_key` and `product_version`.

You populate the new column with a `MERGE` statement that queries the other two columns:

```
=> ALTER TABLE public.product_dimension ADD COLUMN product_ID numeric(8,2);
ALTER TABLE

=> MERGE INTO product_dimension tgt
  USING (SELECT (product_key||'.0'||product_version)::numeric(8,2) AS pid, sku_number
  FROM product_dimension) src
  ON tgt.product_key||'.0'||product_version::numeric=src.pid
  WHEN MATCHED THEN UPDATE SET product_ID = src.pid;
OUTPUT
-----
 60000
(1 row)
```

The following query verifies that the new column values correspond to the values in `product_key` and `product_version`:

```
=> SELECT product_ID, product_key, product_version, product_description
FROM product_dimension
WHERE category_description = 'Medical'
AND product_description ILIKE '%diabetes%'
AND discontinued_flag = 1 ORDER BY product_ID;
product_ID | product_key | product_version | product_description
-----+-----+-----+-----
 5836.02 | 5836 | 2 | Brand #17487 diabetes blood testing kit
14320.02 | 14320 | 2 | Brand #43046 diabetes blood testing kit
18881.01 | 18881 | 1 | Brand #56743 diabetes blood testing kit
(3 rows)
```

MERGE Matching Clauses

`MERGE` supports one instance of the following matching clauses:

- [WHEN MATCHED THEN UPDATE SET](#)
- [WHEN NOT MATCHED THEN INSERT](#)

Each matching clause can specify an additional filter, as described in [Update and Insert Filters](#).

WHEN MATCHED THEN UPDATE SET

Updates all target table rows that are joined to the source table, typically with data from the source table:

```
WHEN MATCHED [ AND update-filter ] THEN UPDATE  
  SET { target-column = expression }[,...]
```

Vertica can execute the join only on unique values in the source table's join column. If the source table's join column contains more than one matching value, the MERGE statement returns with a run-time error.

WHEN NOT MATCHED THEN INSERT

WHEN NOT MATCHED THEN INSERT inserts into the target table a new row for each source table row that is excluded from the join:

```
WHEN NOT MATCHED [ AND insert-filter ] THEN INSERT  
  [ ( column-list ) ] VALUES ( values-list )
```

column-list is a comma-delimited list of one or more target columns in the target table, listed in any order. MERGE maps *column-list* columns to *values-list* values in the same order, and each column-value pair must be [compatible](#). If you omit *column-list*, Vertica maps *values-list* values to columns according to column order in the table definition.

For example, given the following source and target table definitions:

```
CREATE TABLE t1 (a int, b int, c int);  
CREATE TABLE t2 (x int, y int, z int);
```

The following WHEN NOT MATCHED clause implicitly sets the values of the target table columns a, b, and c in the newly inserted rows:

```
MERGE INTO t1 USING t2 ON t1.a=t2.x  
  WHEN NOT MATCHED THEN INSERT VALUES (t2.x, t2.y, t2.z);
```

In contrast, the following WHEN NOT MATCHED clause excludes columns t1.b and t2.y from the merge operation. The WHEN NOT MATCHED clause explicitly pairs two sets of columns from the target and source tables: t1.a to t2.x, and t1.c to t2.z. Vertica sets excluded column t1.b. to null:

```
MERGE INTO t1 USING t2 ON t1.a=t2.x  
  WHEN NOT MATCHED THEN INSERT (a, c) VALUES (t2.x, t2.z);
```

Update and Insert Filters

Each WHEN MATCHED and WHEN NOT MATCHED clause in a MERGE statement can optionally specify an update filter and insert filter, respectively:

```
WHEN MATCHED AND update-filter THEN UPDATE ...  
WHEN NOT MATCHED AND insert-filter THEN INSERT ...
```

Vertica also supports Oracle syntax for specifying update and insert filters:

```
WHEN MATCHED THEN UPDATE SET column-updates WHERE update-filter  
WHEN NOT MATCHED THEN INSERT column-values WHERE insert-filter
```

Each filter can specify multiple conditions. Vertica handles the filters as follows:

- An update filter is applied to the set of matching rows in the target table that are returned by the MERGE join. For each row where the update filter evaluates to true, Vertica updates the specified columns.
- An insert filter is applied to the set of source table rows that are excluded from the MERGE join. For each row where the insert filter evaluates to true, Vertica adds a new row to the target table with the specified values.

For example, given the following data in tables t11 and t22:

```
=> SELECT * from t11 ORDER BY pk;  
pk | col1 | col2 | SKIP_ME_FLAG  
-----  
1 | 2 | 3 | t  
2 | 3 | 4 | t  
3 | 4 | 5 | f  
4 | | 6 | f  
5 | 6 | 7 | t  
6 | | 8 | f  
7 | 8 | | t  
(7 rows)  
  
=> SELECT * FROM t22 ORDER BY pk;  
pk | col1 | col2  
-----  
1 | 2 | 4  
2 | 4 | 8  
3 | 6 |  
4 | 8 | 16  
(4 rows)
```

You can merge data from table t11 into table t22 with the following MERGE statement, which includes update and insert filters:

```
=> MERGE INTO t22 USING t11 ON ( t11.pk=t22.pk )
  WHEN MATCHED
    AND t11.SKIP_ME_FLAG=FALSE AND (
      COALESCE (t22.col1<>t11.col1, (t22.col1 is null)<>(t11.col1 is null))
    )
  THEN UPDATE SET col1=t11.col1, col2=t11.col2
  WHEN NOT MATCHED
    AND t11.SKIP_ME_FLAG=FALSE
  THEN INSERT (pk, col1, col2) VALUES (t11.pk, t11.col1, t11.col2);
OUTPUT
-----
      3
(1 row)

=> SELECT * FROM t22 ORDER BY pk;
pk | col1 | col2
-----+-----+-----
  1 |    2 |    4
  2 |    4 |    8
  3 |    4 |    5
  4 |     |    6
  6 |     |    8
(5 rows)
```

Vertica uses the update and insert filters as follows:

- Evaluates all matching rows against the update filter conditions. Vertica updates each row where the following two conditions both evaluate to true:
 - Source column `t11.SKIP_ME_FLAG` is set to false.
 - The `COALESCE` function evaluates to true.
- Evaluates all non-matching rows in the source table against the insert filter. For each row where column `t11.SKIP_ME_FLAG` is set to false, Vertica inserts a new row in the target table.

MERGE Optimization

You can improve MERGE performance in several ways:

- [Design projections for optimal MERGE performance.](#)
- [Facilitate creation of optimized query plans.](#)
- Use source tables that are smaller than target tables.

Projections for MERGE Operations

The Vertica query optimizer automatically chooses the best projections to implement a merge operation. A good projection design strategy provides projections that help the query optimizer avoid extra sort and data transfer operations, and facilitate MERGE performance.

Tip: You can rely on Database Designer to generate projections that address merge requirements. You can then [customize these projections](#) as needed.

For example, the following MERGE statement fragment joins source and target tables `tgt` and `src`, respectively, on columns `tgt.a` and `src.b`:

```
=> MERGE INTO tgt USING src ON tgt.a = src.b ...
```

Vertica can use a local merge join if projections for tables `tgt` and `src` use one of the following projection designs, where inputs are presorted by projection `ORDER BY` clauses:

- Replicated projections are sorted on:
 - Column `a` for table `tgt`
 - Column `b` for table `src`
- Segmented projections are [identically segmented](#) on:
 - Column `a` for table `tgt`
 - Column `b` for table `src`
 - Corresponding segmented columns

Optimizing MERGE Query Plans

Vertica prepares an optimized query plan if the following conditions are all true:

- The MERGE statement contains both matching clauses `WHEN MATCHED THEN UPDATE SET` and `WHEN NOT MATCHED THEN INSERT`. If the MERGE statement contains only one matching clause, it uses a non-optimized query plan.
- The MERGE statement excludes [update and insert filters](#).

- The target table join column has a unique or primary key constraint. This requirement does not apply to the source table join column.
- Both matching clauses specify all columns in the target table.
- Both matching clauses specify identical source values.

For details on evaluating an [EXPLAIN](#)-generated query plan, see [MERGE Path](#).

The examples that follow use a simple schema to illustrate some of the conditions under which Vertica prepares or does not prepare an optimized query plan for MERGE:

```
CREATE TABLE target(a INT PRIMARY KEY, b INT, c INT) ORDER BY b,a;
CREATE TABLE source(a INT, b INT, c INT) ORDER BY b,a;
INSERT INTO target VALUES(1,2,3);
INSERT INTO target VALUES(2,4,7);
INSERT INTO source VALUES(3,4,5);
INSERT INTO source VALUES(4,6,9);
COMMIT;
```

Optimized MERGE statement

Vertica can prepare an optimized query plan for the following MERGE statement because:

- The target table's join column `t.a` has a primary key constraint.
- All columns in the target table (`a, b, c`) are included in the UPDATE and INSERT clauses.
- The UPDATE and INSERT clauses specify identical source values: `s.a`, `s.b`, and `s.c`.

```
MERGE INTO target t USING source s ON t.a = s.a
  WHEN MATCHED THEN UPDATE SET a=s.a, b=s.b, c=s.c
  WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a, s.b, s.c);
```

```
OUTPUT
-----
2
(1 row)
```

The output value of 2 indicates success and denotes the number of rows updated/inserted from the source into the target.

Non-optimized MERGE statement

In the next example, the MERGE statement runs without optimization because the source values in the UPDATE/INSERT clauses are not identical. Specifically, the UPDATE clause includes constants for columns `s.a` and `s.c` and the INSERT clause does not:

```
MERGE INTO target t USING source s ON t.a = s.a
  WHEN MATCHED THEN UPDATE SET a=s.a + 1, b=s.b, c=s.c - 1
```

```
WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a, s.b, s.c);
```

To make the previous MERGE statement eligible for optimization, rewrite the statement so that the source values in the UPDATE and INSERT clauses are identical:

```
MERGE INTO target t USING source s ON t.a = s.a  
  WHEN MATCHED THEN UPDATE SET a=s.a + 1, b=s.b, c=s.c -1  
  WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a + 1, s.b, s.c - 1);
```

MERGE Restrictions

The following restrictions apply to updating and inserting table data with [MERGE](#).

Constraint Enforcement

MERGE respects all enforced constraints in the target table. If the merge operation attempts to copy values that violate those constraints, MERGE returns with an error and rolls back the merge operation.

Caution: If you run MERGE multiple times using the same target and source table, each iteration is liable to introduce duplicate values into the target columns and return with an error.

Columns Prohibited from Merge

The following columns cannot be specified in a merge operation; attempts to do so return with an error:

- [Identity/auto-increment](#) columns, or columns whose default value is set to a [named sequence](#).
- Vmap columns such as `__raw__` in flex tables.

Removing Table Data

Vertica provides several ways to remove data from a table:

Delete operation	Description
DROP TABLE	Permanently removes a table and its definition, optionally removes associated views and projections.
DELETE	Marks rows with delete vectors and stores them so data can be rolled back to a previous epoch. The data must be purged to reclaim disk space. See Purging Deleted Data .
TRUNCATE TABLE	Removes all storage and history associated with a table. The table structure is preserved for future use. The results of this command cannot be rolled back.
DROP_PARTITION	Removes one partition from a partitioned table. Each partition contains a related subset of data in the table. Partitioned data can be dropped efficiently, and provides query performance benefits. See Using Table Partitions .

Delete Operations Compared

The following table summarizes differences between various delete operations.

Operations and options	Performance	Auto commits	Saves history
DELETE FROM <i>table</i>	Normal	No	Yes
DELETE FROM <i>temp-table</i>	High	No	No
DELETE FROM <i>table</i> <i>where-clause</i>	Normal	No	Yes
DELETE FROM <i>temp-table</i> <i>where-clause</i>	Normal	No	Yes
DELETE FROM <i>temp-table</i> <i>where-clause</i> ON COMMIT PRESERVE ROWS	Normal	No	Yes
DELETE FROM <i>temp-table</i> <i>where-clause</i> ON COMMIT DELETE ROWS	High	Yes	No
DROP <i>table</i>	High	Yes	No
TRUNCATE <i>table</i>	High	Yes	No
TRUNCATE <i>temp-table</i>	High	Yes	No

Operations and options	Performance	Auto commits	Saves history
SELECT DROP_PARTITION (...)	High	Yes	No

Choosing the Best Delete Operation

The following table can help you decide which delete operation is best:

If you want to...	Use...
Delete both table data and definitions and start from scratch.	DROP TABLE
Quickly drop data while preserving table definitions, and reload data.	TRUNCATE TABLE
Regularly perform bulk delete operations.	DROP_PARTITION
Perform occasional small deletes or updates with the option to roll back or review history. See also: Best Practices for DELETE and UPDATE .	DELETE

Best Practices for DELETE and UPDATE

Vertica is optimized for query-intensive workloads, so DELETE and UPDATE queries might not achieve the same level of performance as other queries. DELETE and UPDATE operations go to the WOS by default, but if the data is sufficiently large and would not fit in memory, Vertica automatically switches to using the ROS. See [Using INSERT, UPDATE, and DELETE](#).

The topics that follow discuss best practices when using DELETE and UPDATE operations in Vertica.

DELETE and UPDATE Performance Considerations

To improve the performance of your DELETE and UPDATE queries, consider the following issues:

- **Query performance after large deletes**—A large number of (unpurged) deleted rows can negatively affect query performance.

To eliminate rows that have been deleted from the result, a query must do extra processing. If 10% or more of the total rows in a table have been deleted, the performance of a query on the table degrades. However, your experience may vary depending on the size of the table, the table definition, and the query. If a table has a large number of deleted rows, consider purging those rows to improve performance. For more information on purging, see [Purging Deleted Data](#).

- **Recovery performance**—Recovery is the action required for a cluster to restore K-safety after a crash. Large numbers of deleted records can degrade the performance of a recovery. To improve recovery performance, purge the deleted rows. For more information on purging, see [Purging Deleted Data](#).
- **Concurrency**—DELETE and UPDATE take exclusive locks on the table. Only one DELETE or UPDATE transaction on a table can be in progress at a time and only when no loads (or INSERTs) are in progress. DELETES and UPDATES on different tables can be run concurrently.

For detailed tips about improving DELETE and UPDATE performance, see [DELETE and UPDATE Optimization](#).

Caution: Vertica does not remove deleted data immediately but keeps it as history for the purposes of historical query. A large amount of history can result in slower query

performance. For information about how to configure the appropriate amount of history to retain, see [Purging Deleted Data](#).

DELETE and UPDATE Optimization

The process of optimizing DELETE and UPDATE queries is the same for both operations. Some simple steps can increase the query performance by tens to hundreds of times. The following sections describe several ways to improve projection design and improve DELETE and UPDATE queries to significantly increase DELETE and UPDATE performance.

Note: For large bulk deletion, Vertica recommends using [Partitioned Tables](#) where possible because they provide the best DELETE performance and improve query performance.

Projection Column Requirements for Optimized Deletes

When all columns required by the DELETE or UPDATE predicate are present in a projection, the projection is optimized for DELETES and UPDATES. DELETE and UPDATE operations on such projections are significantly faster than on non-optimized projections.

For example, consider the following table and projections:

```
CREATE TABLE t (a INTEGER, b INTEGER, c INTEGER);  
CREATE PROJECTION p1 (a, b, c) AS SELECT * FROM t ORDER BY a;  
CREATE PROJECTION p2 (a, c) AS SELECT a, c FROM t ORDER BY c, a;
```

In the following query, both p1 and p2 are eligible for DELETE and UPDATE optimization because column a is available:

```
DELETE from t WHERE a = 1;
```

In the following example, only projection p1 is eligible for DELETE and UPDATE optimization because the b column is not available in p2:

```
DELETE from t WHERE b = 1;
```

Optimized Deletes in Subqueries

To be eligible for DELETE optimization, all target table columns referenced in a DELETE or UPDATE statement's WHERE clause must be in the projection definition.

For example, the following simple schema has two tables and three projections:

```
CREATE TABLE tb1 (a INT, b INT, c INT, d INT);  
CREATE TABLE tb2 (g INT, h INT, i INT, j INT);
```

The first projection references all columns in `tb1` and sorts on column `a`:

```
CREATE PROJECTION tb1_p AS SELECT a, b, c, d FROM tb1 ORDER BY a;
```

The buddy projection references and sorts on column `a` in `tb1`:

```
CREATE PROJECTION tb1_p_2 AS SELECT a FROM tb1 ORDER BY a;
```

This projection references all columns in `tb2` and sorts on column `i`:

```
CREATE PROJECTION tb2_p AS SELECT g, h, i, j FROM tb2 ORDER BY i;
```

Consider the following DML statement, which references `tb1.a` in its `WHERE` clause. Since both projections on `tb1` contain column `a`, both are eligible for the optimized `DELETE`:

```
DELETE FROM tb1 WHERE tb1.a IN (SELECT tb2.i FROM tb2);
```

Restrictions

Optimized `DELETE`s are not supported under the following conditions:

- With replicated projections if subqueries reference the target table. For example, the following syntax is not supported:

```
DELETE FROM tb1 WHERE tb1.a IN (SELECT e FROM tb2, tb2 WHERE tb2.e = tb1.e);
```

- With subqueries that do not return multiple rows. For example, the following syntax is not supported:

```
DELETE FROM tb1 WHERE tb1.a = (SELECT k from tb2);
```

Projection Sort Order for Optimizing Deletes

Design your projections so that frequently-used `DELETE` or `UPDATE` predicate columns appear in the sort order of all projections for large `DELETE`s and `UPDATE`s.

For example, suppose most of the `DELETE` queries you perform on a projection look like the following:

```
DELETE from t where time_key < '1-1-2007'
```

To optimize the DELETES, make `time_key` appear in the ORDER BY clause of all your projections. This schema design results in better performance of the DELETE operation.

In addition, add additional sort columns to the sort order such that each combination of the sort key values uniquely identifies a row or a small set of rows. For more information, see [Combine RLE and Sort Order](#). To analyze projections for sort order issues, use the [EVALUATE_DELETE_PERFORMANCE](#) function.

Purging Deleted Data

In Vertica, delete operations do not remove rows from physical storage. [DELETE](#) marks rows as deleted, as does [UPDATE](#), which combines delete and insert operations. In both cases, Vertica retains discarded rows as historical data, which remains accessible to historical queries until it is purged.

The cost of retaining historical data is twofold:

- Disk space is allocated to deleted rows and delete markers.
- Typical (non-historical) queries must read and skip over deleted data, which can impact performance.

A purge operation permanently removes historical data from physical storage and frees disk space for reuse. Only historical data that precedes the Ancient History Mark (AHM) is eligible to be purged.

You can purge data in two ways:

- [Set a purge policy](#).
- [Manually purge data](#).

In both cases, Vertica purges all historical data up to and including the AHM epoch and resets the AHM.

Caution: Large delete and purge operations can take a long time to complete, so use them sparingly. If your application requires deleting data on a regular basis, such as by month or year, consider designing tables that take advantage of [table partitioning](#). If partitioning is not suitable, consider [rebuilding the table](#).

Setting a Purge Policy

The preferred method for purging data is to establish a policy that determines which deleted data is eligible to be purged. Eligible data is automatically purged when the Tuple Mover performs mergeout operations.

Vertica provides two methods for determining when deleted data is eligible to be purged:

- Specifying the time for which delete data is saved
- Specifying the number of epochs that are saved

Specifying the Time for Which Delete Data Is Saved

Specifying the time for which delete data is saved is the preferred method for determining which deleted data can be purged. By default, Vertica saves historical data only when nodes are down.

To change the specified time for saving deleted data, use the `HistoryRetentionTime` [configuration parameter](#):

```
=> ALTER DATABASE mydb SET HistoryRetentionTime = {seconds | -1};
```

In the above syntax:

- `seconds` is the amount of time (in seconds) for which to save deleted data.
- `-1` indicates that you do not want to use the `HistoryRetentionTime` configuration parameter to determine which deleted data is eligible to be purged. Use this setting if you prefer to use the other method (`HistoryRetentionEpochs`) for determining which deleted data can be purged.

The following example sets the history epoch retention level to 240 seconds:

```
=> ALTER DATABASE mydb SET HistoryRetentionTime = 240;
```

Specifying the Number of Epochs That Are Saved

Unless you have a reason to limit the number of epochs, Micro Focus recommends that you specify the time over which delete data is saved.

To specify the number of historical epoch to save through the `HistoryRetentionEpochs` configuration parameter:

1. Turn off the `HistoryRetentionTime` configuration parameter:

```
=> ALTER DATABASE mydb SET HistoryRetentionTime = -1;
```

2. Set the history epoch retention level through the `HistoryRetentionEpochs` configuration parameter:

```
=> ALTER DATABASE mydb SET HistoryRetentionEpochs = {num_epochs | -1};
```

- `num_epochs` is the number of historical epochs to save.
- `-1` indicates that you do not want to use the `HistoryRetentionEpochs` configuration parameter to trim historical epochs from the epoch map. By default, `HistoryRetentionEpochs` is set to `-1`.

The following example sets the number of historical epochs to save to 40:

```
=> ALTER DATABASE mydb SET HistoryRetentionEpochs = 40;
```

Modifications are immediately implemented across all nodes within the database cluster. You do not need to restart the database.

Note: If both `HistoryRetentionTime` and `HistoryRetentionEpochs` are specified, `HistoryRetentionTime` takes precedence.

See [Epoch Management Parameters](#) for additional details.

Disabling Purge

If you want to preserve all historical data, set the value of both historical epoch retention parameters to `-1`, as follows:

```
=> ALTER DATABASE mydb SET HistoryRetentionTime = -1;  
=> ALTER DATABASE mydb SET HistoryRetentionEpochs = -1;
```

Manually Purging Data

You manually purge deleted data as follows:

1. Determine the point in time to which you want to purge deleted data.
2. Set the ancient history mark (AHM) to this point in time using one of the following functions:
 - `SET_AHM_TIME` sets the AHM to the epoch that includes the specified `TIMESTAMP` value on the initiator node.
 - `SET_AHM_EPOCH` sets the AHM to the specified epoch.

- [GET_AHM_TIME](#) returns a `TIMESTAMP` value representing the AHM.
- [GET_AHM_EPOCH](#) returns the number of the epoch in which the AHM is located.
- [MAKE_AHM_NOW](#) sets the AHM to the greatest allowable value. This lets you purge all deleted data.

When you use `SET_AHM_TIME` or `GET_AHM_TIME`, keep in mind that the timestamp you specify is mapped to an epoch, which by default has a three-minute granularity. Thus, if you specify an AHM time of `2008-01-01 00:00:00.00`, the resulting purge might permanently remove as much as the first three minutes of 2008, or fail to remove the last three minutes of 2007.

Note: The system prevents you from setting the AHM beyond the point where it prevents recovery in the event of node failure.

3. Purge deleted data from the desired projections with one of the following functions:
 - [PURGE](#) purges all projections in the physical schema.
 - [PURGE_TABLE](#) purges all projections anchored to the specified table.
 - [PURGE_PROJECTION](#) purges the specified projection.
 - [PURGE_PARTITION](#) purges a specified partition.
4. The tuple mover performs a mergeout operation to purge the data. Vertica periodically invokes the tuple mover to perform mergeout operations, as configured by [tuple mover parameters](#). You can manually invoke the tuple mover by calling the function `DO_TM_TASK`.

Caution: Manual purge operations can take a long time.

Truncating Tables

`TRUNCATE TABLE` removes all storage associated with the target table and its projections. Vertica preserves the table and the projection definitions. If the truncated table has out-of-date projections, those projections are cleared and marked up-to-date when `TRUNCATE TABLE` returns.

`TRUNCATE TABLE` commits the entire transaction after statement execution, even if truncating the table fails. You cannot roll back a `TRUNCATE TABLE` statement.

Use `TRUNCATE TABLE` for testing purposes. You can use it to remove all data from a table and load it with fresh data, without recreating the table and its projections.

Table Locking

`TRUNCATE TABLE` takes an O (owner) lock on the table until the truncation process completes. The savepoint is then released. If the operation cannot obtain an [O lock](#) on the target table, Vertica tries to close any internal [tuple mover](#) (TM) sessions that are running on that table. If successful, the operation can proceed. Explicit TM operations that are running in user sessions do not close. If an explicit TM operation is running on the table, the operation proceeds only when the TM operation is complete.

Restrictions

- You cannot truncate an external table.
- You can truncate the fact table of a pre-join projection. However, you cannot truncate the dimension table of the same pre-join projection. If a table acts as a dimension table in any pre-join projections, you must drop those projections before you can execute `TRUNCATE TABLE` on the dimension table.

Examples

```
=> INSERT INTO sample_table (a) VALUES (3);
=> SELECT * FROM sample_table;
a
---
3
(1 row)

=> TRUNCATE TABLE sample_table;
TRUNCATE TABLE
=> SELECT * FROM sample_table;
a
---
(0 rows)
```

Rebuilding Tables

You can reclaim disk space on a large scale by rebuilding tables, as follows:

1. Create a table with the same (or similar) definition as the table to rebuild.
2. Create projections for the new table.
3. Copy data from the target table into the new one with `INSERT . . SELECT`.
4. Drop the old table and its projections.

Note: Rather than dropping the old table, you can rename it and use it as a backup copy. Before doing so, verify that you have sufficient disk space for both the new and old tables.

5. Rename the new table with `ALTER TABLE . . RENAME`, using the name of the old table.

Caution: When you rebuild a table, Vertica purges the table of all delete vectors that precede the AHM. This prevents historical queries on any older epoch.

Projection Considerations

- You must have enough disk space to contain the old and new projections at the same time. If necessary, you can drop some of the old projections before loading the new table. You must, however, retain at least one superprojection of the old table (or two buddy superprojections to maintain K-safety) until the new table is loaded. (See [Prepare Disk Storage Locations](#) in Installing Vertica for disk space requirements.)
- You can specify different names for the new projections or use `ALTER TABLE . . RENAME` to change the names of the old projections.
- The relationship between tables and projections does not depend on object names. Instead, it depends on object identifiers that are not affected by rename operations. Thus, if you rename a table, its projections continue to work normally.

Dropping Tables

`DROP TABLE` drops a table from the database catalog. If any projections are associated with the table, `DROP TABLE` returns an error message unless it also includes the `CASCADE` option. One exception applies: the table only has an [auto-generated superprojection](#) (auto-projection) associated with it.

Using CASCADE

In the following example, `DROP TABLE` tries to remove a table that has several projections associated with it. Because it omits the `CASCADE` option, Vertica returns an error:

```
=> DROP TABLE d1;
NOTICE: Constraint - depends on Table d1
NOTICE: Projection d1p1 depends on Table d1
NOTICE: Projection d1p2 depends on Table d1
NOTICE: Projection d1p3 depends on Table d1
NOTICE: Projection f1d1p1 depends on Table d1
NOTICE: Projection f1d1p2 depends on Table d1
NOTICE: Projection f1d1p3 depends on Table d1
ERROR: DROP failed due to dependencies: Cannot drop Table d1 because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
=> DROP TABLE d1 CASCADE;
DROP TABLE
=> CREATE TABLE mytable (a INT, b VARCHAR(256));
CREATE TABLE
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

The next attempt includes the `CASCADE` option and succeeds:

```
=> DROP TABLE d1 CASCADE;
DROP TABLE
=> CREATE TABLE mytable (a INT, b VARCHAR(256));
CREATE TABLE
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

Using IF EXISTS

In the following example, `DROP TABLE` includes the option `IF EXISTS`. This option specifies not to report an error if one or more of the tables to drop does not exist. This clause is useful in SQL scripts—for example, to ensure that a table is dropped before you try to recreate it:

```
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Table doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

Dropping and Restoring View Tables

Views that reference a table that is dropped and then replaced by another table with the same name continue to function and use the contents of the new table. The new table must have the same column definitions.

Managing Client Connections

You can limit the number of active concurrent sessions a user can open to the server. Limiting user sessions lets you better manage sessions and prevents excessive server communication.

You can define connection limits at three levels: user level, for an individual node or entire cluster.

You can use the following parameters to close a session that has been idle for a period of time:

- `IDLESESSIONTIMEOUT`: Sets how much time can elapse before a session times out.
- `DEFAULTIDLESESSIONTIMEOUT`— A configuration parameter that sets the default timeout value for users who do not have `idlesessiontimeout` set. See [Security Parameters](#).

The `IDLESESSIONTIMEOUT` parameter applies only to sessions that are idle, not to sessions where query execution is in progress. An idle session is one that has no queries running. If a client is slow or unresponsive during query execution, that time does not apply to timeout. For example, if you are doing a streaming batch insert, the time that it takes to perform this operation is not counted towards timeout.

Use Cases

A user executes a query, and for some reason the query takes an unusually long time to finish (for example, because of server traffic or query complexity). In this case, the user may mistakenly think the query failed and opens another session to run the same query. Now, two sessions are running the same query, using an extra connection.

To prevent this situation, limit the number of sessions a user can run by creating or modifying a user and setting `MAXCONNECTIONS`. For example:

```
=> CREATE USER u1 MAXCONNECTIONS 10 ON NODE;
```

Another issue setting client connections prevents is when a user connects to the server many times. Too many user connections exhausts the number of allowable connections set with the `MaxClientSessions` parameter (see [Managing Sessions](#)).

Note: No user can have a `MAXCONNECTIONS` limit greater than the `MaxClientSessions` setting.

Manage Client Sessions

After setting a user's connection limits, you can manage the client sessions by monitoring changes to the user's connection parameters. Also be aware that behavior changes can occur with client connection limits when a node:

- Is removed
- Is added
- Goes down
- Comes up

Client Connection Changes

When you modify a user's client connection parameters, be aware that:

- Changing a user's MAXCONNECTIONS setting only affects new connections.
- Changing a user's connection mode from DATABASE to NODE does not affect current sessions. All new sessions are reserved on the invoking node rather than the entire cluster.
- Changing a user's connection mode from NODE to DATABASE does not affect current sessions. New connection requests are reserved on the entire cluster.

Node Going Up or Down

In terms of honoring connection limits, no significant impact exists when nodes go down or come up in between connection requests. No special actions are needed to handle this.

However, note the following:

- If a node goes down, its active session exits and other nodes in the cluster also drop their sessions. This frees up connections. The query may hang in which case the blocked sessions are reasonable and as expected.

User-Defined Session Timeout

As the DBADMIN, you can set a user's idle session timeout with IDLESESSIONTIMEOUT. A user can edit the session timeout value as follows:

```
=> SET SESSION IDLESESSIONTIMEOUT 'interval';
```

In the user role, you can only lower the IDLESESSIONTIMEOUT value. If you attempt to increase the value, the following error message appears:

```
ERROR 0: New idlesessiontimeout 00:12 would exceed user limit of 00:10
```

Run the following query to see the current IDLESESSIONTIMEOUT value:

```
=> SHOW IDLESESSIONTIMEOUT;
name | idlesessiontimeout
-----+-----
user1 | 00:08
```

Set the DEFAULTIDLESESSIONTIMEOUT configuration parameter as the default session timeout value:

```
=> ALTER DATABASE <dbname> SET DEFAULTIDLESESSIONTIMEOUT = interval;
```

This value applies to users who do not have an IDLESESSIONTIMEOUT value set.

In the user role, you can lower the idle timeout for a specific session as follows:

```
=> SET SESSION IDLESESSIONTIMEOUT 'interval';
```

Close a Session for a User

If necessary, you can manually close a user session with [CLOSE_USER_SESSIONS](#):

```
=> SELECT CLOSE_USER_SESSIONS ('user-name');
          close_user_sessions
-----
Session close command sent. Check v_monitor.session for progress
```

Set Client Connection Limits

Setting a user's client connection limits allows you to control how many simultaneous sessions a user can run. This prevents Vertica from exhausting the allowable concurrent connections set by `MaxClientSessions` and improves the usability of the system.

MaxClientSessions Limits

Client connection limits cannot be higher than the limits set in `MaxClientSessions`. Therefore:

- If you need to increase a user's allowed concurrent connections, check to see if `MaxClientSessions` needs to be increased first.
- If you decrease `MaxClientSessions`, the database uses the lesser of the two connection values between `MaxClientSessions` and the user-level connection limit set with `MAXCONNECTIONS`.

Set Connection Limits for a User

Set connection limits for a user as follows:

```
=> CREATE USER user1 MAXCONNECTIONS 10 ON NODE IDLESESSIONTIMEOUT '10 mins';
```

This example limits `user1` to ten concurrent connections on an individual node. An idle session for the user times out after ten minutes. For more information see [CREATE USER](#).

You can also modify an existing user to add connection limits with [ALTER USER](#).

```
=> ALTER USER user2 MAXCONNECTIONS 5 ON DATABASE;
```

View User Connection Limits and Session Timeouts

See the [USERS](#) table to determine if a user has connection limits and session timeouts implemented.

```
=> SELECT user_name, max_connections, connection_limit_mode, idle_session_timeout FROM users;  
user_name | max_connections | connection_limit_mode | idle_session_timeout
```

max1	10	node	00:10
max2	5	database	unlimited

IDLESESSIONTIMEOUT

When setting the IDLESESSIONTIMEOUT value, be aware that a system is considered idle when it is waiting for instructions from the client.

The system starts tracking the timeout value from the point the server starts waiting for any type of message. For example, if the server receives a parse and is waiting for a bind, the time spent waiting is considered idle time and IDLESESSIONTIMEOUT starts.

When you change a user's IDLESESSIONTIMEOUT value, the new value affects only those sessions started after the update.

Working with Projections

Unlike traditional databases that store data in tables, Vertica physically stores table data in projections, which are collections of table columns.

Projections store data in a format that optimizes query execution. Similar to materialized views, they store result sets on disk rather than compute them each time they are used in a query. Vertica automatically refreshes these result sets with updated or new data.

Projections can be generally classified in two ways:

- [How projection data is distributed](#) on the cluster. A projection can be defined to divide its data into multiple segments, or maintain all projection data as a single unsegmented unit.
- [How much data a projection contains, and the nature of that data](#). For example, each table Vertica requires a superprojection, which contains all table columns. You can also create query-specific projections, which contain only the subset of table columns to process a given query.

For more general information about Vertica projections, see [Physical Schema](#) in Vertica Concepts.

Projection Naming

Vertica identifies projections according to the following conventions, where *proj-basename* is the name assigned to this projection by [CREATE PROJECTION](#).

Unsegmented Projections

Unsegmented projections conform to the following naming conventions:

<i>table-name_</i> super	Identifies the auto projection that Vertica automatically creates when you initially load data into an unsegmented table, where <i>table-basename</i> is the table name specified in CREATE TABLE . The auto projection is always a superprojection. For example: <pre>store.customer_dimension_super</pre>
-----------------------------	---

<i>proj-basename</i> [_unseg]	<p>Identifies an unsegmented projection. If <i>proj-basename</i> is identical to the anchor table name, Vertica appends the string <code>_unseg</code> to the projection name. If the projection is copied on all nodes, this projection name maps to all instances.</p> <p>For example:</p> <pre>store.customer_dimension_unseg</pre>
----------------------------------	--

Segmented Projections

Segmented projections conform to the following naming convention:

<i>proj-basename_</i> <i>boffset</i>	<p>Identifies buddy projections for a segmented projection, where <i>offset</i> identifies the projection's node location relative to all other buddy projections. All buddy projections share the same project base name.</p> <p>For example:</p> <pre>store.store_orders_fact_b0 store.store_orders_fact_b1</pre> <p>One exception applies: Vertica uses the following convention to name live aggregate projections:</p> <ul style="list-style-type: none">• <i>proj-basename</i>• <i>proj-basename_b1</i>• ...
---	--

Projections of Copied Tables

Vertica creates projections for tables that you create from existing tables with [CREATE TABLE LIKE...INCLUDING PROJECTIONS](#).

The following table describes the algorithm that Vertica uses to assign base names to the new projections:

Source projection name	Target		
	Schema	Table	Projection base name
<i>string</i>	Same	Different	<i>tgtTableName_string</i>
<i>string</i>	Different	Same	<i>string</i>
<i>string</i>	Different	Different	<i>tgtTableName_string</i>
<i>srcTableName_string</i>	Same	Different	<i>tgtTableName_string</i>
<i>srcTableName_string</i>	Different	Different	<i>tgtTableName_string</i>

For example, given the following table and its segmented projections:

Table name	Projection names
public.T1	T1_b0 T1_b1
	T1_proj1_b0 T1_proj1_b1
	proj2_b0 proj2_b1

The following CREATE TABLE . . . LIKE statements copy table public.T1 and its projections to tables public.T2 and private.T1:

```
CREATE TABLE public.T2 LIKE public.T1 INCLUDING PROJECTIONS;
CREATE TABLE private.T1 LIKE public.T1 INCLUDING PROJECTIONS;
```

For each new table, Vertica copies the public.T1 projections as follows:

New table name	Copied projection names
public.T2	T2_b0 T2_b1
	T2_proj1_b0 T2_proj1_b1
	T2_proj2_b0 T2_proj2_b1
private.T1	T1_b0 T1_b1
	T1_proj1_b0 T1_proj1_b1
	proj2_b0 proj2_b1

Auto-Projections

Auto-projections are superprojections that Vertica automatically generates for tables, both temporary and persistent. The following rules apply to all auto-projections:

- Vertica creates the auto-projection in the same schema as the table.
- Auto-projections conform to encoding, sort order, segmentation, and K-safety as specified in the table's creation statement.
- If the table creation statement contains an `AS SELECT` clause, Vertica uses some properties of the projection definition's underlying query.

Auto-Projection Triggers

The conditions for creating auto-projections differ, depending on whether the table is temporary or persistent:

Table type	Auto-projection trigger
Temporary	<code>CREATE TEMPORARY TABLE</code> statement unless it includes <code>NO PROJECTION</code> .
Persistent	<code>CREATE TABLE</code> statement contains one of these clauses: <ul style="list-style-type: none">• <code>AS SELECT</code>• <code>ENCODED BY</code>• <code>ORDER BY</code>• <code>SEGMENTED BY / UNSEGMENTED</code>• <code>KSAFE</code> If none of these conditions is true, Vertica automatically creates a superprojection (if one does not already exist) only when you first load data into the table with <code>INSERT</code> or <code>COPY</code> .

Default Segmentation and Sort Order

If `CREATE TABLE` or `CREATE TEMPORARY TABLE` omits a segmentation (`SEGMENTED BY` or `UNSEGMENTED`) or `ORDER BY` clause, Vertica segments and sorts auto-projections according to the table's manner of creation:

- If `CREATE [TEMPORARY] TABLE` contains an `AS SELECT` clause, and the query output is segmented, the auto-projection uses the same segmentation. If the result set is already sorted, the projection uses the same sort order.
- In all other cases, Vertica evaluates table column constraints to determine how to sort and segment the projection, as shown below:

Constraints	Sorted by:	Segmented by:
Primary key	Primary key	Primary key
Primary and foreign keys	<ol style="list-style-type: none"> 1. Foreign keys 2. Primary key 	Primary key
Foreign keys only	<ol style="list-style-type: none"> 1. Foreign keys 2. Remaining columns excluding <code>LONG data types</code>, in the order specified by <code>CREATE TABLE</code>. 	<p>All columns excluding <code>LONG data types</code>, up to the limit set in configuration parameter <code>MaxAutoSegColumns</code> (by default 32).</p> <p>Vertica orders segmentation according to data type size, as follows:</p>
None	All columns excluding <code>LONG data types</code> , in the order specified by <code>CREATE TABLE</code> .	<ol style="list-style-type: none"> 1. Small (< 8 byte) data type columns 2. Large (>8 byte) data type columns

Unsegmented Projections

In many cases, dimension tables are relatively small, so you do not need to segment them. Accordingly, you should design a K-safe database so projections for its dimension tables are

replicated without segmentation on all cluster nodes. You create unsegmented projections with a `CREATE PROJECTION` statement that includes the clause `UNSEGMENTED ALL NODES`. This clause specifies to create identical instances of the projection on all cluster nodes.

The following example shows how to create an unsegmented projection for the table `store.store_dimension`:

```
=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state)
      AS SELECT store_key, store_name, store_city, store_state
      FROM store.store_dimension
      UNSEGMENTED ALL NODES;
CREATE PROJECTION
```

Vertica uses the same name to identify all instances of the unsegmented projection—in this example, `store.store_dimension_proj`. The keyword `ALL NODES` specifies to replicate the projection on all nodes:

```
=> \dj store.store_dimension_proj
      List of projections
Schema | Name | Owner | Node | Comment
-----+-----+-----+-----+-----
store | store_dimension_proj | dbadmin | v_vmart_node0001 |
store | store_dimension_proj | dbadmin | v_vmart_node0002 |
store | store_dimension_proj | dbadmin | v_vmart_node0003 |
(3 rows)
```

For more information about projection name conventions, see [Projection Naming](#).

Segmented Projections

You typically create segmented projections for large fact tables. Vertica splits segmented projections into chunks (segments) of similar size and distributes these segments evenly across the cluster. System K-safety determines how many duplicates (*buddies*) of each segment are created and maintained on different nodes.

You create segmented projections with a `CREATE PROJECTION` statement that includes a `SEGMENTED BY clause`.

The following `CREATE PROJECTION` statement creates projection `public.employee_dimension_super`. It specifies to include all columns in table `public.employee_dimension`. The hash segmentation clause invokes the Vertica `HASH` function to segment projection data on the column `employee_key`; it also includes the `ALL NODES` clause, which specifies to distribute projection data evenly across all nodes in the cluster:

```
=> CREATE PROJECTION public.employee_dimension_super
      AS SELECT * FROM public.employee_dimension
      ORDER BY employee_key
```

```
SEGMENTED BY hash(employee_key) ALL NODES;
```

If the database is K-safe, Vertica creates multiple buddies for this projection and distributes them on different nodes across the cluster. In this case, database K-safety is set to 1, so Vertica creates two buddies for this projection. It uses the projection name `employee_dimension_super` as the basename for the two buddy identifiers it creates—in this example, `employee_dimension_super_b0` and `employee_dimension_super_b1`:

```
=> SELECT projection_name FROM projections WHERE projection_basename='employee_dimension_super';
      projection_name
-----
employee_dimension_super_b0
employee_dimension_super_b1
(2 rows)
```

K-Safe Database Projections

K-safety is implemented differently for segmented and unsegmented projections, as described below. Examples assume database K-safety is set to 1 in a 3-node database, and uses projections for two tables:

- `store.store_orders_fact` is a large fact table. The projection for this table should be segmented. Vertica distributes projection segments uniformly across the cluster.
- `store.store_dimension` is a smaller dimension table. The projection for this table should be unsegmented. Vertica copies a complete instance of this projection on each cluster node.

Segmented Projections

In a K-safe database, the database requires K+1 instances, or buddies, of each projection segment. For example, if database K-safety is set to 1, the database requires two instances, or buddies, of each projection segment.

You can set K-safety on individual segmented projections through the [CREATE PROJECTION](#) option `KSAFE`. Projection K-safety must be equal to or greater than database K-safety. If you omit setting `KSAFE`, the projection obtains K-safety from the database.

The following [CREATE PROJECTION](#) defines a segmented projection for the fact table `store.store_orders_fact`:

```
=> CREATE PROJECTION store.store_orders_fact
      (prodkey, ordernum, storekey, total)
      AS SELECT product_key, order_number, store_key, quantity_ordered*unit_price
      FROM store.store_orders_fact
      SEGMENTED BY HASH(product_key, order_number) ALL NODES KSAFE 1;
CREATE PROJECTION
```

The following keywords in the `CREATE PROJECTION` statement pertain to setting projection K-safety:

<code>SEGMENTED BY</code>	Specifies how to segment projection data for distribution across the cluster. In this example, the segmentation expression specifies Vertica's built-in HASH function.
<code>ALL NODES</code>	Specifies to distribute projection segments across all cluster nodes.
<code>K-SAFE 1</code>	Sets K-safety to 1. Vertica creates two projection buddies with these identifiers: <ul style="list-style-type: none">• <code>store.store_orders_fact_b0</code>• <code>store.store_orders_fact_b1</code>

Unsegmented Projections

In a K-safe database, unsegmented projections must be replicated on all nodes. Thus, the `CREATE PROJECTION` statement for an unsegmented projection must include the segmentation clause `UNSEGMENTED ALL NODES`. This instructs Vertica to create identical instances (buddies) of the projection on all cluster nodes. If you create an unsegmented projection on a single node, Vertica regards it unsafe and does not use it.

The following example shows how to create an unsegmented projection for the table `store.store_dimension`:

```
=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state)
      AS SELECT store_key, store_name, store_city, store_state
      FROM store.store_dimension
      UNSEGMENTED ALL NODES;
CREATE PROJECTION
```

Vertica uses the same name to identify all instances of the unsegmented projection—in this example, `store.store_dimension_proj`. The keyword `ALL NODES` specifies to replicate the projection on all nodes:

```
=> \dj store.store_dimension_proj
List of projections
Schema | Name | Owner | Node | Comment
-----+-----+-----+-----+-----
store | store_dimension_proj | dbadmin | v_vmart_node0001 |
store | store_dimension_proj | dbadmin | v_vmart_node0002 |
store | store_dimension_proj | dbadmin | v_vmart_node0003 |
(3 rows)
```

For more information about projection name conventions, see [Projection Naming](#).

Refreshing Projections

[CREATE PROJECTION](#) does not load data into physical storage. If the anchor tables already contain data, run [START_REFRESH](#) to update the projection. Depending on how much data is in the tables, updating a projection can be time consuming. When a projection is up-to-date, however, it is updated automatically as part of [COPY](#), [DELETE](#), [INSERT](#), [MERGE](#), or [UPDATE](#) statements.

Monitoring Projection Refresh on Buddy Projections

Vertica refreshes projections only after you create a buddy projection. If you run [START_REFRESH](#) after creating a projection, Vertica returns this message:

```
Starting refresh background process
```

However, the refresh begins only after you create buddy projections.

To confirm whether a refresh operation is complete, review the `vertica.log` file. [GET_PROJECTIONS](#) also provides the final status of the projection refresh:

```
=> SELECT GET_PROJECTIONS('customer_dimension');
GET_PROJECTIONS
-----
Current system K is 1.
# of Nodes: 3.
Table public.customer_dimension has 2 projections.

Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate] [Stats]
-----
public.customer_dimension_unseg [Segmented: No] [Seg Cols: ] [K: 2] [public.customer_dimension_unseg]
[Safe: Yes] [UptoDate: Yes] [Stats: Yes]
```

```
public.customer_dimension_DBD_1_rep_VMartDesign_node0001 [Segmented: No] [Seg Cols: ] [K: 2]  
[public.customer_dimension_DBD_1_rep_VMartDesign_node0001] [Safe: Yes] [UptoDate: Yes] [Stats: Yes]
```

(1 row)

Dropping Projections

Projections can be dropped explicitly through the [DROP PROJECTION](#) statement. They are also implicitly dropped when you drop their anchor table.

Using Table Partitions

Data partitioning is [defined as a table property](#), and is implemented on all projections of that table. On all load, refresh, and recovery operations, the Vertica Tuple Mover automatically partitions data into separate ROS containers. Each ROS container contains data for a single partition; depending on space requirements, a partition can span multiple ROS containers.

For example, it is common to partition data by time slices. If a table contains decades of data, you can partition it by year. If the table contains only one year of data, you can partition it by month.

Logical divisions of data can significantly improve query execution. For example, if you query a table on a column that is in the table's partition clause, the query optimizer can quickly isolate the relevant ROS containers (see [Eliminating Partitions](#)).

Partitions can also facilitate DML operations. For example, given a table that is partitioned by months, you might drop all data for the oldest month when a new month begins. In this case, Vertica can easily identify the ROS containers that store the partition data to drop. For further details on partitions and using them, see this article, [FAQ: Vertica Partitions](#).

Defining Partitions

You can specify partitioning for a table when you initially define the table with [CREATE TABLE](#). Alternatively, you can specify partitioning for an existing table by modifying its definition with [ALTER TABLE](#). In the first case, Vertica automatically partitions data with each load operation. In the second case, you must explicitly repartition existing data with the Vertica function [PARTITION_TABLE](#).

A table definition specifies partitioning through a `PARTITION BY` clause:

```
PARTITION BY expression
```

where *expression* resolves to a value derived from one or more table columns.

Partitioning a New Table

The following `CREATE TABLE` statement creates the `trade` table, which partitions data into separate years:

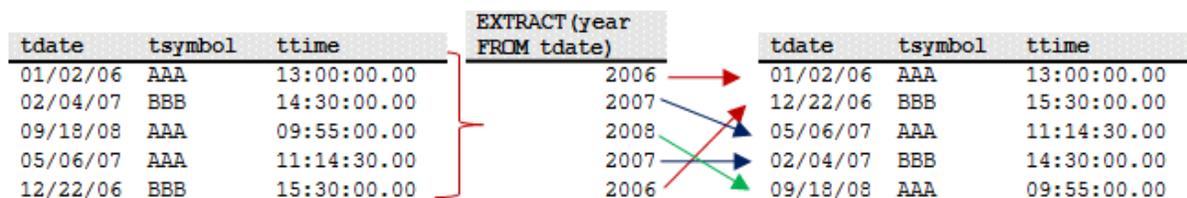
```
=> CREATE TABLE trade (  
    tdate DATE NOT NULL,
```

```

    tsymbol VARCHAR(8) NOT NULL,
    ttime TIME)
PARTITION BY EXTRACT (year FROM tdate);
CREATE TABLE
CREATE PROJECTION trade_p (tdate, tsymbol, ttime) AS
SELECT * FROM trade
ORDER BY tdate, tsymbol, ttime UNSEGMENTED ALL NODES;
INSERT INTO trade VALUES ('01/02/08' , 'AAA' , '13:00:00');
INSERT INTO trade VALUES ('02/04/09' , 'BBB' , '14:30:00');
INSERT INTO trade VALUES ('09/18/10' , 'AAA' , '09:55:00');
INSERT INTO trade VALUES ('05/06/09' , 'AAA' , '11:14:30');
INSERT INTO trade VALUES ('12/22/08' , 'BBB' , '15:30:00');

```

As new data is inserted into the table, Vertica implements its `PARTITION BY` clause as follows:



Partitioning an Existing Table

Use the `ALTER TABLE` statement to partition or repartition an existing table, as specified by the `PARTITION BY` clause. On executing this statement, Vertica immediately drops any existing partition keys.

For example, the following statements create the unpartitioned `trade` table, alter the table to add a constraint, and then insert data:

```

=> CREATE TABLE public.employee_dim(
    employee_key int NOT NULL,
    employee_gender varchar(8),
    courtesy_title varchar(8),
    employee_first_name varchar(64),
    employee_middle_initial varchar(8),
    employee_last_name varchar(64),
    employee_age int,
    hire_date date,
    employee_street_address varchar(256),
    employee_city varchar(64) NOT NULL,
    employee_state char(2),
    employee_region char(32) NOT NULL,
    job_title varchar(64),
    reports_to int,
    salaried_flag int,
    annual_salary int,
    hourly_rate float,

```

```
vacation_days int
);
CREATE TABLE
=> ALTER TABLE public.employee_dim ADD CONSTRAINT C_PRIMARY PRIMARY KEY (employee_key) ENABLED;
ALTER TABLE
=> INSERT INTO employee_dim SELECT * from employee_dimension;
OUTPUT
-----
10000
(1 row)
```

You can view how Vertica distributes the newly loaded data in various ROS containers across the cluster nodes, by querying the system table [STORAGE_CONTAINERS](#):

```
=> SELECT node_name, projection_name, storage_type, total_row_count FROM storage_containers
WHERE projection_name ilike '%employee_dim_b%' ORDER BY projection_name;
node_name      | projection_name | storage_type | total_row_count
-----|-----|-----|-----
v_vmart_node0001 | employee_dim_b0 | ROS          | 3347
v_vmart_node0002 | employee_dim_b0 | ROS          | 3321
v_vmart_node0003 | employee_dim_b0 | ROS          | 3332
v_vmart_node0001 | employee_dim_b1 | ROS          | 3332
v_vmart_node0002 | employee_dim_b1 | ROS          | 3347
v_vmart_node0003 | employee_dim_b1 | ROS          | 3321
(6 rows)
```

The following statement partitions the table on column `employee_region`:

```
=> ALTER TABLE employee_dim PARTITION BY employee_region;
=> NOTICE 4954: The new partitioning scheme will produce 6 partitions
WARNING 4493: Queries using table "employee_dim" may not perform optimally since the data may not be
repartitioned in accordance with the new partition expression
HINT: Use "ALTER TABLE public.employee_dim REORGANIZE;" to repartition the data
ALTER TABLE
```

The partition clause omits the keyword `REORGANIZE`, so table data remains unpartitioned. If you query the `STORAGE_CONTAINERS` table again, Vertica returns the same or similar results as before. Vertica repartitions table data only when it executes a mergeout operation. You can initiate a mergeout with functions [DO_TM_TASK](#) or [PARTITION_TABLE](#), or with an `ALTER TABLE . . . REORGANIZE` statement:

```
=> SELECT PARTITION_TABLE('employee_dim');
PARTITION_TABLE
-----
Task: partition operation
(Table: public.employee_dim) (Projection: public.employee_dim_b0)
(Table: public.employee_dim) (Projection: public.employee_dim_b1)
(1 row)
```

Now when you query the `STORAGE_CONTAINERS` table, the results confirm that `employee_dim` data is repartitioned into logical divisions among ROS containers. The following query joins `STORAGE_CONTAINERS` with system table [PARTITIONS](#). The truncated output below shows

how Vertica redistributes the repartitioned data. The data is now ordered by partition keys (East, Midwest, etc.) within each projection buddy (employee_dim_b0 and employee_dim_b1):

```
=> SELECT c.projection_name,
       p.partition_key,
       c.node_name,
       c.storage_type,
       c.total_row_count ROS_rows
FROM STORAGE_CONTAINERS c JOIN PARTITIONS p ON c.projection_name=p.projection_name
WHERE c.projection_name LIKE '%employee_dim%' ORDER BY c.projection_name, p.partition_key;
```

projection_name	partition_key	node_name	storage_type	ROS_rows
employee_dim_b0	East	v_vmart_node0002	ROS	745
employee_dim_b0	East	v_vmart_node0002	ROS	557
...				
employee_dim_b0	East	v_vmart_node0003	ROS	761
employee_dim_b0	East	v_vmart_node0003	ROS	563
...				
employee_dim_b0	East	v_vmart_node0001	ROS	366
employee_dim_b0	East	v_vmart_node0001	ROS	920
employee_dim_b0	MidWest	v_vmart_node0002	ROS	745
employee_dim_b0	MidWest	v_vmart_node0002	ROS	557
...				
employee_dim_b0	MidWest	v_vmart_node0003	ROS	761
employee_dim_b0	MidWest	v_vmart_node0003	ROS	563
...				
employee_dim_b0	MidWest	v_vmart_node0001	ROS	366
employee_dim_b0	MidWest	v_vmart_node0001	ROS	920
employee_dim_b0	NorthWest	v_vmart_node0002	ROS	745
employee_dim_b0	NorthWest	v_vmart_node0002	ROS	557
...				
employee_dim_b0	NorthWest	v_vmart_node0003	ROS	761
employee_dim_b0	NorthWest	v_vmart_node0003	ROS	563
...				
employee_dim_b0	NorthWest	v_vmart_node0001	ROS	366
employee_dim_b0	NorthWest	v_vmart_node0001	ROS	920
employee_dim_b0	South	v_vmart_node0002	ROS	745
employee_dim_b0	South	v_vmart_node0002	ROS	557
...				
employee_dim_b0	South	v_vmart_node0002	ROS	632
employee_dim_b0	South	v_vmart_node0002	ROS	95
employee_dim_b0	South	v_vmart_node0003	ROS	761
employee_dim_b0	South	v_vmart_node0003	ROS	563
...				
employee_dim_b0	South	v_vmart_node0001	ROS	366
employee_dim_b0	South	v_vmart_node0001	ROS	920
employee_dim_b0	SouthWest	v_vmart_node0002	ROS	745
employee_dim_b0	SouthWest	v_vmart_node0002	ROS	557
...				
employee_dim_b0	SouthWest	v_vmart_node0003	ROS	761
employee_dim_b0	SouthWest	v_vmart_node0003	ROS	563
...				
employee_dim_b0	SouthWest	v_vmart_node0001	ROS	366
employee_dim_b0	SouthWest	v_vmart_node0001	ROS	920
employee_dim_b0	West	v_vmart_node0002	ROS	745
employee_dim_b0	West	v_vmart_node0002	ROS	557
...				

```

employee_dim_b0 | West          | v_vmart_node0003 | ROS          | 761
employee_dim_b0 | West          | v_vmart_node0003 | ROS          | 563
...
employee_dim_b0 | West          | v_vmart_node0001 | ROS          | 366
employee_dim_b0 | West          | v_vmart_node0001 | ROS          | 920
employee_dim_b1 | East          | v_vmart_node0001 | ROS          | 350
employee_dim_b1 | East          | v_vmart_node0001 | ROS          | 350
...
employee_dim_b1 | East          | v_vmart_node0003 | ROS          | 954
employee_dim_b1 | East          | v_vmart_node0001 | ROS          | 921
employee_dim_b1 | MidWest      | v_vmart_node0001 | ROS          | 350
employee_dim_b1 | MidWest      | v_vmart_node0001 | ROS          | 350
...
employee_dim_b1 | MidWest      | v_vmart_node0003 | ROS          | 954
employee_dim_b1 | MidWest      | v_vmart_node0001 | ROS          | 921
employee_dim_b1 | NorthWest    | v_vmart_node0001 | ROS          | 350
employee_dim_b1 | NorthWest    | v_vmart_node0001 | ROS          | 350
...
employee_dim_b1 | NorthWest    | v_vmart_node0003 | ROS          | 954
employee_dim_b1 | NorthWest    | v_vmart_node0001 | ROS          | 921
employee_dim_b1 | South        | v_vmart_node0001 | ROS          | 350
employee_dim_b1 | South        | v_vmart_node0001 | ROS          | 350
...
employee_dim_b1 | South        | v_vmart_node0003 | ROS          | 954
employee_dim_b1 | South        | v_vmart_node0001 | ROS          | 921
employee_dim_b1 | SouthWest    | v_vmart_node0001 | ROS          | 350
employee_dim_b1 | SouthWest    | v_vmart_node0001 | ROS          | 350
...
employee_dim_b1 | SouthWest    | v_vmart_node0003 | ROS          | 954
employee_dim_b1 | SouthWest    | v_vmart_node0001 | ROS          | 921
employee_dim_b1 | West         | v_vmart_node0001 | ROS          | 350
employee_dim_b1 | West         | v_vmart_node0001 | ROS          | 350
...
employee_dim_b1 | West         | v_vmart_node0003 | ROS          | 954
employee_dim_b1 | West         | v_vmart_node0001 | ROS          | 921
(648 rows)

```

Caution: If you repartition a table without specifying REORGANIZE, Vertica stores new data according to the new partition expression, while existing data storage remains unchanged. Until you explicitly partition all table data, using either with ALTER TABLE . . . REORGANIZE statement, or with PARTITION_TABLE, query performance, DROP_PARTITION, and node recovery can be adversely affected.

Partitioning Best Practices

Recommendations

Minimize number of partitions

Vertica supports up to 1024 ROS containers per partition. In general, a ROS contains only one partition (although a partition can [sometimes span multiple ROS containers](#)). Each partition typically spans multiple ROS containers. Be aware that any delete operation requires Vertica to open all ROS containers, so a large number of partitions can adversely affect performance.

In practice, it is highly unlikely that you will approach this maximum. For optimal performance, Vertica recommends that the number of partitions range between 10 and 20, and not exceed more than 50. This range is typically compatible with most business requirements.

Avoid partitioning by LONG data types

For maximum performance, do not partition table data on LONG VARBINARY and LONG VARCHAR columns.

Tip: Partitioning by Year and Month

To partition by both year and month, the partition clause should pad the month out to two digits with an expression like this:

```
PARTITION BY EXTRACT(year FROM tdate)*100 + EXTRACT(month FROM tdate)
```

This expression formats partition keys as follows:

```
201101  
201102  
201103  
...  
201111  
201112
```

Comparing Partitioning and Segmentation

In Vertica, partitioning and segmentation are separate concepts and achieve different goals to localize data:

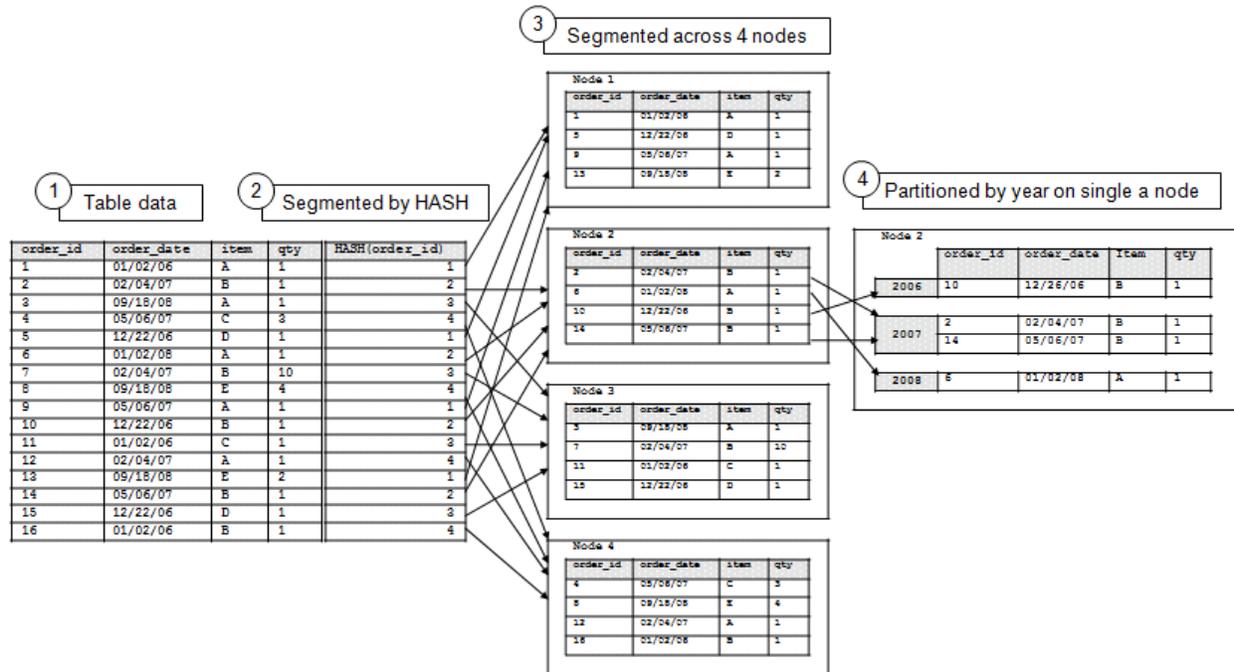
- **Segmentation** refers to organizing and distributing data across cluster nodes for fast data purges and query performance. Segmentation aims to distribute data evenly across multiple database nodes so all nodes participate in query execution. You specify segmentation with the `CREATE PROJECTION` statement's `hash segmentation clause`.
- **Partitioning** specifies how to organize data within individual nodes for distributed computing. Node partitions let you easily identify data you wish to drop and help reclaim disk space. You specify partitioning with the `CREATE TABLE` statement's `PARTITION BY` clause.

For example: partitioning data by year makes sense for retaining and dropping annual data. However, segmenting the same data by year would be inefficient, because the node holding data for the current year would likely answer far more queries than the other nodes.

The following diagram illustrates the flow of segmentation and partitioning on a four-node database cluster:

1. Example table data
2. Data segmented by `HASH(order_id)`
3. Data segmented by hash across four nodes
4. Data partitioned by year on a single node

While partitioning occurs on all four nodes, the illustration shows partitioned data on one node for simplicity.



See Also

- [Reclaiming Disk Space From Deleted Table Data](#)
- [Identical Segmentation](#)
- [Projection Segmentation](#)
- [CREATE PROJECTION](#)
- [CREATE TABLE](#)

Managing Partitions

Vertica provides various options to let you manage and monitor the partitions you create.

PARTITION_TABLE Function

The function [PARTITION_TABLE](#) physically separates partitions into separate containers. Only ROS containers with more than one distinct value participate in the split.

The following example creates a simple table states and partitions the data by state:

```
=> CREATE TABLE states (year INTEGER NOT NULL, state VARCHAR NOT NULL) PARTITION BY state;  
=> CREATE PROJECTION states_p (state, year) AS  
    SELECT * FROM states ORDER BY state, year UNSEGMENTED ALL NODES;
```

Run `PARTITION_TABLE` to partition the table `states`:

```
=> SELECT PARTITION_TABLE('states');  
           PARTITION_TABLE  
-----  
Task: partition operation  
(Table: public.states) (Projection: public.states_p)  
(1 row)
```

PARTITIONS System Table

You can display partition metadata, one row per partition key, per ROS container, by querying the [PARTITIONS](#) system table.

Given the unsegmented projection `states_p` replicated across three nodes, the following query on the `PARTITIONS` table returns twelve rows, representing twelve ROS containers:

```
=> SELECT PARTITION_KEY, ROS_ID, ROS_SIZE_BYTES, ROS_ROW_COUNT, NODE_NAME FROM partitions WHERE  
PROJECTION_NAME='states_p' order by ROS_ID;  
PARTITION_KEY | ROS_ID | ROS_SIZE_BYTES | ROS_ROW_COUNT | NODE_NAME  
-----  
VT | 45035996281231297 | 95 | 14 | v_vmart_node0001  
PA | 45035996281231309 | 92 | 11 | v_vmart_node0001  
NY | 45035996281231321 | 90 | 9 | v_vmart_node0001  
MA | 45035996281231333 | 96 | 15 | v_vmart_node0001  
VT | 49539595902704977 | 95 | 14 | v_vmart_node0002  
PA | 49539595902704989 | 92 | 11 | v_vmart_node0002  
NY | 49539595902705001 | 90 | 9 | v_vmart_node0002  
MA | 49539595902705013 | 96 | 15 | v_vmart_node0002  
VT | 54043195530075651 | 95 | 14 | v_vmart_node0003  
PA | 54043195530075663 | 92 | 11 | v_vmart_node0003  
NY | 54043195530075675 | 90 | 9 | v_vmart_node0003  
MA | 54043195530075687 | 96 | 15 | v_vmart_node0003  
(12 rows)
```

Restrictions

You cannot use volatile functions in a `PARTITION BY` expression. For example, the value of `TIMESTAMP_TZ` depends on user settings.

Eliminating Partitions

If the ROS containers of partitioned tables are not needed, Vertica can eliminate the containers from being processed during query execution. To eliminate ROS containers, Vertica compares query predicates to partition-related metadata.

Each ROS partition expression column maintains the minimum and maximum values of data stored in that ROS, and Vertica uses those min/max values to potentially eliminate ROS containers from query planning. Partitions that cannot contain matching values are not scanned. For example, if a ROS does not contain data that satisfies a given query predicate, the optimizer eliminates (prunes) that ROS from the query plan. After non-participating ROS containers have been eliminated, queries that use partitioned tables run more quickly.

Note: Partition pruning occurs at query run time and requires a query predicate on the partitioning column.

Assume a table that is partitioned by year (2007, 2008, 2009) into three ROS containers, one for each year. Given the following series of commands, the two ROS containers that contain data for 2007 and 2008 fall outside the boundaries of the requested year (2009) and get eliminated.

```
=> CREATE TABLE ... PARTITION BY EXTRACT(year FROM date);  
=> SELECT ... WHERE date = '12-2-2009';
```

date	amount	date	amount	date	amount
11/11/09	6	03/13/08	96	07/12/07	43
06/05/09	12	04/21/08	17	03/02/07	45
...
12/02/09	8	12/02/08	7	12/02/07	68
Min: 01/01/09 Max: 12/31/09		Min: 01/01/08 Max: 12/31/08		Min: 01/01/07 Max: 12/31/07	

Making Past Partitions Eligible for Elimination

The following procedure lets you make past partitions eligible for elimination. The easiest way to guarantee that all ROS containers are eligible is to:

1. Create a fact table with the same projections as the existing table.
2. Use `INSERT . . SELECT` to populate the new table.
3. Drop the original table and rename the new table.

If the disk lacks enough space for a second copy of the fact table, follow this procedure:

1. Verify that the Tuple Mover finished all post-upgrade work—for example, when the following command shows no mergeout activity:

```
=> SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

2. Identify which partitions need to be merged to get the ROS minimum/maximum values:

```
=> SELECT DISTINCT table_schema, projection_name, partition_key  
FROM partitions p LEFT OUTER JOIN vs_ros_min_max_values v  
ON p.ros_id = v.delid  
WHERE v.min_value IS null;
```

3. Insert a record into each partition that has ineligible ROS containers, and commit.
4. Delete each inserted record and commit again.

At this point, the Tuple Mover automatically merges ROS containers from past partitions.

Verifying the ROS Merge

1. Query the `TUPLE_MOVER_OPERATIONS` table:

```
=> SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

2. Check for any partitions that need to be merged:

```
=> SELECT DISTINCT table_schema, projection_name, partition_key  
FROM partitions p LEFT OUTER JOIN vs_ros_min_max_values v  
ON p.ros_id = v.rosid  
WHERE v.min_value IS null;
```

Examples

Assume a table that is partitioned by time and will use queries that restrict data on time.

```
=> CREATE TABLE time ( tdate DATE NOT NULL, tnum INTEGER)
    PARTITION BY EXTRACT(year FROM tdate);
=> CREATE PROJECTION time_p (tdate, tnum) AS
=> SELECT * FROM time ORDER BY tdate, tnum UNSEGMENTED ALL NODES;
```

Note: Projection sort order has no effect on partition elimination.

```
=> INSERT INTO time VALUES ('03/15/04' , 1);
=> INSERT INTO time VALUES ('03/15/05' , 2);
=> INSERT INTO time VALUES ('03/15/06' , 3);
=> INSERT INTO time VALUES ('03/15/06' , 4);
```

The data inserted in the previous series of commands are loaded into three ROS containers, one per year, as that is how the data is partitioned:

```
=> SELECT * FROM time ORDER BY tnum;
  tdate  | tnum
-----+-----
2004-03-15 |    1  --ROS1 (min 03/01/04, max 03/15/04)
2005-03-15 |    2  --ROS2 (min 03/15/05, max 03/15/05)
2006-03-15 |    3  --ROS3 (min 03/15/06, max 03/15/06)
2006-03-15 |    4  --ROS3 (min 03/15/06, max 03/15/06)
(4 rows)
```

Here's what happens when you query the `time` table:

- In this query, Vertica can eliminate ROS2 because it is only looking for year 2004:

```
=> SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004';
```

- In the next query, Vertica can eliminate both ROS1 and ROS3:

```
=> SELECT COUNT(*) FROM time WHERE tdate = '10/07/2005';
```

- The following query has an additional predicate on the `tnum` column for which no minimum/maximum values are maintained. In addition, the use of logical operator OR is not supported, so no ROS elimination occurs:

```
=> SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004' OR tnum = 7;
```

Dropping Partitions

Use the [DROP_PARTITION](#) function to drop a partition. Typically, this is a fast operation that discards all ROS containers that contain data for the partition. If the WOS contains table data,

DROP_PARTITION first forces a moveout operation.

Tip: When a ROS container has data for a single partition, you can discard that storage location with [DROP_LOCATION](#) after dropping that partition.

Dropping Partitions Due to Space Constraints

If your hardware has fixed disk space, you might need to configure a regular process to roll out old data by dropping partitions.

For example, if you have only enough space to store data for a fixed number of days, configure Vertica to drop the partition with the oldest date. To do so, create a time-based job scheduler, such as `cron`, to drop the partition on a regular basis during low-load periods.

If the ingest rate for data has peaks and valleys, you can use two techniques to manage how you drop partitions:

- Set up a process to check the disk space on a regular (daily) basis. If the percentage of used disk space exceeds a certain threshold—for example, 80%—drop the oldest partition.
- Add an artificial column in a partition that increments based on a metric like row count. For example, that column might increment each time that the row count increases by 100 rows. Set up a process that queries that column on a regular (daily) basis. If the value in the new column exceeds a certain threshold—for example, 100—drop the oldest partition, and set the column value back to 0.

Table Locks

DROP_PARTITION acquires an exclusive [O lock](#) on the target table to block any DML operation (DELETE, UPDATE, INSERT, or COPY) that might affect table data. The lock also blocks SELECT statements that are issued at [SERIALIZABLE](#) isolation level.

If the operation cannot obtain an [O lock](#) on the target table, Vertica tries to close any internal [tuple mover](#) (TM) sessions that are running on that table. If successful, the operation can proceed. Explicit TM operations that are running in user sessions do not close. If an explicit TM operation is running on the table, the operation proceeds only when the TM operation is complete.

Examples

Using the example schema in [Defining Partitions](#), the following command explicitly drops the 2009 partition key from table `trade`:

```
SELECT DROP_PARTITION('trade', 2009);
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

Here, the partition key is specified:

```
SELECT DROP_PARTITION('trade', EXTRACT('year' FROM '2009-01-01'::date));
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

The following example creates a table called `dates` and partitions the table by year:

```
CREATE TABLE dates (year INTEGER NOT NULL,
                    month VARCHAR(8) NOT NULL)
PARTITION BY year * 12 + month;
```

The following statement drops the partition using a constant for Oct 2010 ($2010*12 + 10 = 24130$):

```
SELECT DROP_PARTITION('dates', '24130');
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

Alternatively, the expression can be placed in line: `SELECT DROP_PARTITION('dates', 2010*12 + 10);`

The following command first reorganizes the data if it is unpartitioned and then explicitly drops the 2009 partition key from table `trade`:

```
SELECT DROP_PARTITION('trade', 2009, false, true);
      DROP_PARTITION
-----
Partition dropped
(1 row)
```

Archiving Partitions

You can move partitions from one table to another with the Vertica function [MOVE_PARTITIONS_TO_TABLE](#). This function is useful for archiving old partitions, as part of the following procedure:

1. [Identify the partitions to archive, and move them to a temporary staging table](#) with `MOVE_PARTITIONS_TO_TABLE`.
2. [Back up the staging table](#).
3. [Drop the staging table](#).

You can retrieve and restore archived partitions at any time, as described in [Restoring Archived Partitions](#).

For general information about moving partitions, see [MOVE_PARTITIONS_TO_TABLE](#).

Move Partitions to Staging Tables

You archive historical data by identifying the partitions you wish to remove from a table. You then move each partition (or group of partitions) to a temporary staging table.

Before calling `MOVE_PARTITIONS_TO_TABLE`, you must:

- Drop any pre-join projections associated with the source table.
- Refresh all out-of-date projections.

The following recommendations apply to staging tables:

- To facilitate the backup process, create a unique schema for the staging table of each archiving operation.
- Specify new names for staging tables. This ensures that they do not contain partitions from previous move operations.

If the table does not exist, Vertica creates a table from the source table's definition, by calling `CREATE TABLE` with `LIKE` and `INCLUDING PROJECTIONS` clause. The new table inherits ownership from the source table. For details, see [Replicating a Table](#).

- Use staging names that enable other users to easily identify partition contents. For example, if a table is partitioned by dates, use a name that specifies a date or date range.

In the following example, `MOVE_PARTITIONS_TO_TABLE` specifies to move a single partition to the staging table `partn_backup.tradfes_200801`.

```
=> SELECT MOVE_PARTITIONS_TO_TABLE (  
      'prod_trades',  
      '200801',  
      '200801',  
      'partn_backup.tradfes_200801');  
MOVE_PARTITIONS_TO_TABLE  
-----  
1 distinct partition values moved at epoch 15.  
(1 row)
```

Back Up the Staging Table

After you create a staging table, you archive it through an object-level backup using a `vbr` configuration file. For detailed information, see [Backing Up and Restoring the Database](#).

Important: Vertica recommends performing a full database backup before the object-level backup, as a precaution against data loss. You can only restore object-level backups to the original database.

Drop the Staging Tables

After the backup is complete, you can drop the staging table as described in [Dropping Tables](#).

Restoring Archived Partitions

You can restore partitions that you previously moved to an intermediate table, archived as an object-level backup, and then dropped.

Note: Restoring an archived partition requires that the original table definition has not changed since the partition was archived and dropped. If you have changed the table definition, you can only restore an archived partition using `INSERT/SELECT` statements, which are not described here.

These are the steps to restoring archived partitions:

1. Restore the backup of the intermediate table you saved when you moved one or more partitions to archive (see [Archiving Partitions](#)).
2. Move the restored partitions from the intermediate table to the original table.
3. Drop the intermediate table.

Swapping Partitions

`SWAP_PARTITIONS_BETWEEN_TABLES` combines the operations of `DROP_PARTITION` and `MOVE_PARTITIONS_TO_TABLE` as a single transaction. `SWAP_PARTITIONS_BETWEEN_TABLES` is useful if you regularly load partitioned data from one table into another and need to refresh partitions in the second table.

For example, you might have a table of revenue that is partitioned by date, and you routinely move data into it from a staging table. Occasionally, the staging table contains data for dates that are already in the target table. In this case, you must first remove partitions from the target table for those dates, then replace them with the corresponding partitions from the staging table. You can accomplish both tasks with a single call to `SWAP_PARTITIONS_BETWEEN_TABLES`.

By wrapping the drop and move operations within a single transaction, `SWAP_PARTITIONS_BETWEEN_TABLES` maintains integrity of the swapped data. If any task in the swap operation fails, the entire operation fails and is rolled back.

Examples

In the following example, `SWAP_PARTITIONS_BETWEEN_TABLES` drops from table `member_info` all partitions in range specified by partition keys `2008` and `2009`. It replaces the dropped partitions with the corresponding partitions in source table `customer_info`:

```
=> SELECT SWAP_PARTITIONS_BETWEEN_TABLES('customer_info',2008,2009,'member_info');
          SWAP_PARTITIONS_BETWEEN_TABLES
-----
1 partition values from table customer_info and 2 partition values from table
member_info are swapped at epoch 1250.
```

See Also

[Tutorial for Swapping Partitions](#)

Tutorial for Swapping Partitions

The following example shows how to create two partitioned tables and then swap certain partitions between the tables.

Both tables have the same definition and have partitions for various year values. You swap the partitions where `year = 2008` and `year = 2009`. Both tables have at least two rows that will be swapped.

1. Create the `customer_info` table:

```
=> CREATE TABLE customer_info (  
    customer_id INT PRIMARY KEY NOT NULL,  
    first_name VARCHAR(25),  
    last_name VARCHAR(35),  
    city VARCHAR(25),  
    year INT NOT NULL)  
ORDER BY last_name  
PARTITION BY year;
```

2. Insert data into the `customer_info` table:

```
=> INSERT INTO customer_info VALUES (1, 'Joe', 'Smith', 'Denver', 2008);  
=> INSERT INTO customer_info VALUES (2, 'Bob', 'Jones', 'Boston', 2008);  
=> INSERT INTO customer_info VALUES (3, 'Silke', 'Muller', 'Frankfurt', 2007);  
=> INSERT INTO customer_info VALUES (4, 'Simone', 'Bernard', 'Paris', 2014);  
=> INSERT INTO customer_info VALUES (5, 'Vijay', 'Kumar', 'New Delhi', 2010);
```

3. View the table data:

```
=> SELECT * FROM customer_info;  
customer_id | first_name | last_name | city    | year  
-----+-----+-----+-----+-----  
1 | Joe      | Smith    | Denver  | 2008  
2 | Bob      | Jones    | Boston  | 2008  
3 | Silke    | Muller   | Frankfurt | 2007  
4 | Simone   | Bernard  | Paris   | 2014  
5 | Vijay    | Kumar    | New Delhi | 2010
```

4. Create a second table, `member_info`, that has the same definition as `customer_info`:

```
=> CREATE TABLE member_info (  
    customer_id INT PRIMARY KEY NOT NULL,  
    first_name VARCHAR(25),  
    last_name VARCHAR(35),  
    city VARCHAR(25),  
    year INT NOT NULL)  
ORDER BY last_name  
PARTITION BY year;
```

5. Insert data into the member_info table:

```
=> INSERT INTO member_info VALUES (1, 'Jane', 'Doe', 'Miami', 2001);  
=> INSERT INTO member_info VALUES (2, 'Mike', 'Brown', 'Chicago', 2014);  
=> INSERT INTO member_info VALUES (3, 'Patrick', 'OMalley', 'Dublin', 2008);  
=> INSERT INTO member_info VALUES (4, 'Ana', 'Lopez', 'Madrid', 2009);  
=> INSERT INTO member_info VALUES (5, 'Mike', 'Green', 'New York', 2008);
```

6. View the data in the member_info table:

```
=> SELECT * FROM member_info;  
customer_id | first_name | last_name | city | year  
-----+-----+-----+-----+-----  
1 | Jane      | Doe      | Miami | 2001  
2 | Mike     | Brown   | Chicago | 2014  
3 | Patrick  | OMalley | Dublin | 2008  
4 | Ana     | Lopez  | Madrid | 2009  
5 | Mike     | Green   | New York | 2008
```

7. To swap the partitions, run the SWAP_PARTITIONS_BETWEEN_TABLES function:

```
=> SELECT SWAP_PARTITIONS_BETWEEN_TABLES('customer_info',2008,2009,'member_info');
```

8. To verify that the partitions have been swapped, query the contents of both tables, and confirm the following results:

- After the swap, the rows in both tables whose year values are 2008 or 2009 have moved to the other table.
- The partitions that contain the rows for Ana Lopez, Mike Green, and Patrick OMalley moved from member_info to customer_info.
- The partition that contains the rows for Joe Smith and Bob Jones moved from customer_info to member_info.

```
=> SELECT * FROM customer_info;  
customer_id | first_name | last_name | city | year  
-----+-----+-----+-----+-----
```

```
4 | Simone | Bernard | Paris | 2014
5 | Vijay | Kumar | New Delhi | 2010
3 | Silke | Muller | Frankfurt | 2007
4 | Ana | Lopez | Madrid | 2009
5 | Mike | Green | New York | 2008
3 | Patrick | OMalley | Dublin | 2008

=> SELECT * FROM member_info;
customer_id | first_name | last_name | city | year
-----+-----+-----+-----+-----
2 | Bob | Jones | Boston | 2008
1 | Joe | Smith | Denver | 2008
2 | Mike | Brown | Chicago | 2014
1 | Jane | Doe | Miami | 2001
```

Handling Multiple Partitions in One ROS Container

Occasionally, a ROS container contains rows that belong to more than one partition. For example, in some cases refresh and recovery operations can generate ROS containers with mixed partitions. In general, Vertica segregates data from different partitions in different ROS containers, but exceptions can occur. This is generally true only in a database that was created with an early version of Vertica.

If a ROS container has data from multiple partitions, [DROP_PARTITION](#) executes as follows:

1. If the ROS contains table data, forces a moveout operation.
2. Divides the ROS container data among two containers:
 - One container holds the data that belongs to the partition to drop.
 - The second container holds the remaining partitions
3. Drops the specified partition.

About Constraints

Constraints specify rules on what values can go into a column. Examples of constraints:

- Primary key
- Foreign key
- Unique
- Check
- Not NULL

Using constraints can help you maintain data integrity in one or more columns. Do not define constraints on columns unless you expect to keep the data consistent.

Vertica can use constraints to perform optimizations (such as the optimized MERGE) that assume the data is consistent.

Adding Constraints

Add constraints on one or more table columns using the following SQL commands:

- [CREATE TABLE](#): Add a constraint on one or more columns.
- [ALTER TABLE](#): Add or drop a constraint on one or more columns.

Vertica recommends naming a constraint but it is optional; if you specify the `CONSTRAINT` keyword, you must give a name for the constraint.

The examples that follow illustrate several ways of adding constraints. For additional details, see:

- [Primary Key Constraints](#)
- [Foreign Key Constraints](#)
- [Unique Constraints](#)
- [Not NULL Constraints](#)
- [Check constraints](#)

Adding Column Constraints with CREATE TABLE

There are several ways to add a constraint on a column using `CREATE TABLE`:

- On the column definition using the `CONSTRAINT` keyword, which requires that you assign a constraint name, in this example, `dim1PK`:

```
CREATE TABLE dim1 ( c1 INTEGER CONSTRAINT dim1PK PRIMARY KEY,  
                    c2 INTEGER  
                    );
```

- On the column definition, omitting the `CONSTRAINT` keyword. When you omit the `CONSTRAINT` keyword, you cannot specify a constraint name:

```
CREATE TABLE dim1 ( c1 INTEGER PRIMARY KEY,  
                    c2 INTEGER
```

```
);
```

- After the column definition, using the `CONSTRAINT` keyword and assigning a name, in this example, `dim1PK`:

```
CREATE TABLE dim1 ( c1 INTEGER,  
                    c2 INTEGER,  
                    CONSTRAINT dim1pk PRIMARY KEY(c1)  
);
```

- After the column definition, omitting the `CONSTRAINT` keyword:

```
CREATE TABLE dim1 ( c1 INTEGER,  
                    c2 INTEGER,  
                    PRIMARY KEY(c1)  
);
```

Adding Two Constraints on a Column

To add more than one constraint on a column, specify the constraints one after another when you create the table column. For example, the following statement enforces both not NULL and unique constraints on the `customer_key` column, indicating that the column values cannot be NULL and must be unique:

```
CREATE TABLE test1 ( id INTEGER NOT NULL UNIQUE,  
                    ...  
);
```

Adding a Foreign Key Constraint on a Column

There are four ways to add a foreign key constraint on a column using `CREATE TABLE`. The `FOREIGN KEY` keywords are not valid *on* the column definition, only *after* the column definition:

- On the column definition, use the `CONSTRAINT` and `REFERENCES` keywords and name the constraint, in this example, `fact1dim1PK`. This example creates a column with a named foreign key constraint referencing the table (`dim1`) with the primary key (`c1`):

```
CREATE TABLE fact1 ( c1 INTEGER CONSTRAINT fact1dim1FK REFERENCES dim1(c1),  
                    c2 INTEGER  
);
```

- On the column definition, omit the **CONSTRAINT** keyword and use the **REFERENCES** keyword with the table name and column:

```
CREATE TABLE fact1 ( c1 INTEGER REFERENCES dim1(c1),  
c2 INTEGER  
);
```

- After the column definition, use the **CONSTRAINT**, **FOREIGN KEY**, and **REFERENCES** keywords and name the constraint:

```
CREATE TABLE fact1 ( c1 INTEGER,  
c2 INTEGER,  
CONSTRAINT fk1 FOREIGN KEY(c1) REFERENCES dim1(c1)  
);
```

- After the column definition, omitting the **CONSTRAINT** keyword:

```
CREATE TABLE fact1 ( c1 INTEGER,  
c2 INTEGER,  
FOREIGN KEY(c1) REFERENCES dim1(c1)  
);
```

Each of the following **ALTER TABLE** statements adds a foreign key constraint on an existing column, with and without using the **CONSTRAINT** keyword:

```
ALTER TABLE fact2  
ADD CONSTRAINT fk1 FOREIGN KEY (c1) REFERENCES dim2(c1);
```

or

```
ALTER TABLE fact2 ADD FOREIGN KEY (c1) REFERENCES dim2(c1);
```

For additional details, see [Foreign Key Constraints](#).

Adding Multicolumn Constraints

The following example defines a primary key constraint on multiple columns by first defining the table columns (**c1** and **c2**), and then specifying both columns in a **PRIMARY KEY** clause:

```
CREATE TABLE dim ( c1 INTEGER,  
c2 INTEGER,  
PRIMARY KEY (c1, c2)  
);
```

To specify multicolumn (compound) primary keys, the following example uses CREATE TABLE to define the columns. After creating the table, ALTER TABLE defines the compound primary key and names it dim2PK:

```
CREATE TABLE dim2 ( c1 INTEGER,  
  c2 INTEGER,  
  c3 INTEGER NOT NULL,  
  c4 INTEGER UNIQUE  
);  
ALTER TABLE dim2  
  ADD CONSTRAINT dim2PK PRIMARY KEY (c1, c2);
```

In the next example, you define a compound primary key as part of the CREATE TABLE statement. Then you specify the matching foreign key constraint to table dim2 using CREATE TABLE and ALTER TABLE:

```
CREATE TABLE dim2 ( c1 INTEGER,  
  c2 INTEGER,  
  c3 INTEGER NOT NULL,  
  c4 INTEGER UNIQUE,  
  PRIMARY KEY (c1, c2)  
);  
CREATE TABLE fact2 (  
  c1 INTEGER,  
  c2 INTEGER,  
  c3 INTEGER NOT NULL,  
  c4 INTEGER UNIQUE  
);  
ALTER TABLE fact2  
  ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2(c1, c2);
```

Specify a foreign key constraint using a reference to the table that contains the primary key. In the ADD CONSTRAINT clause, the REFERENCES column names are optional. The following ALTER TABLE statement is equivalent to the previous ALTER TABLE statement:

```
ALTER TABLE fact2 ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2;
```

Adding Constraints on Tables with Existing Data

When you add a constraint on a column with existing data, Vertica:

- Verifies the validity of the column values only if you are adding a primary key, unique key, or check constraint enabled for automatic enforcement.
- Does *not* verify the validity of column values for other constraint types.

If your data does not conform to the declared non-enabled constraints, your queries could yield unexpected results.

Use [ANALYZE_CONSTRAINTS](#) to check for constraint violations in your column. If you find violations, use the ALTER COLUMN SET/DROP parameters of the [ALTER TABLE](#) statement to apply or remove a constraint on an existing column.

Note: You can configure your system to automatically enforce primary key, unique key and check constraints during DML. For information on automatic enforcement, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Altering Column Constraints

The following example uses [ALTER TABLE](#) to add column b with not NULL and default 5 constraints to a table test6:

```
CREATE TABLE test6 (a INT);  
ALTER TABLE test6 ADD COLUMN b INT DEFAULT 5 NOT NULL;
```

Use ALTER TABLE with the ALTER COLUMN and SET NOT NULL clauses to add the constraint on column a in table test6:

```
ALTER TABLE test6 ALTER COLUMN a SET NOT NULL;
```

Use the SET NOT NULL or DROP NOT NULL clause to add or remove a not NULL column constraint:

```
=> ALTER TABLE T1 ALTER COLUMN x SET NOT NULL;  
=> ALTER TABLE T1 ALTER COLUMN x DROP NOT NULL;
```

Use these clauses so that the column has the proper constraints when you have added or removed a primary key constraint on a column. You can also use them any time you want to add or remove the NOT NULL constraint.

Note: A PRIMARY KEY constraint includes a NOT NULL constraint. However, if you drop the PRIMARY KEY constraint on a column, the NOT NULL constraint remains on that column.

Enforcing Constraints

Check constraints are enforced by default unless you disable individual constraints when you create or alter the constraint, or set the parameter EnableNewCheckConstraintsByDefault to 0 (disabled). See [check constraints](#) for more information.

To maximize query performance, Vertica checks for primary key and foreign key violations when loading into the fact table of a pre-join projection. For more details, see [Enforcing Primary Key and Foreign Key Constraints](#).

Vertica checks for not NULL constraint violations when loading data, but it does not check for unique constraint violations for constraints that are not enabled.

To validate table data on constraints that are not enabled, load data without committing it by using the [COPY](#) with the NO COMMIT option. Then perform a post-load check using the [ANALYZE_CONSTRAINTS](#) function. If constraint violations are found, you can roll back the load because you have not committed it. For more details, see [Detecting Constraint Violations with ANALYZE_CONSTRAINTS](#).

Note: Vertica enforces check constraints automatically by default. You can also enforce primary key and unique key constraints automatically. See [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

See Also

- [ALTER TABLE](#)
- [CREATE TABLE](#)
- [COPY](#)
- [ANALYZE_CONSTRAINTS](#)

Primary Key Constraints

A primary key (PK) is a single column or combination of columns (called a *compound key*) that uniquely identifies each row in a table. A primary key constraint contains unique, non-null values.

When you apply the primary key constraint, the NOT NULL and unique constraints are added implicitly. You do not need to specify them when you create the column. However, if you remove the primary key constraint, the NOT NULL constraint continues to apply to the column. To remove the NOT NULL constraint after removing the primary key constraint, use the ALTER COLUMN DROP NOT NULL parameter of the [ALTER TABLE](#) statement (see [Dropping Constraints](#)).

The following example shows how you can add a primary key constraint on the `employee_id` field:

```
CREATE TABLE employees (employee_id INTEGER PRIMARY KEY);
```

Alternatively, you can add a primary key constraint after the column is created:

```
CREATE TABLE employees (employee_id INTEGER);  
ALTER TABLE employees  
  ADD PRIMARY KEY (employee_id);
```

Note: If you specify a primary key constraint using ALTER TABLE, the system returns the following message, which is informational only. The primary key constraint is added to the designated column.

```
WARNING 2623: Column "employee_id" definition changed to NOT NULL
```

You can also use primary keys to constrain more than one column:

```
CREATE TABLE employees (employee_id INTEGER,  
  employee_gender CHAR(1),  
  PRIMARY KEY (employee_id, employee_gender)  
);
```

When you enable automatic enforcement of primary or unique key constraints, Vertica applies enforcement for:

- INSERT
- UPDATE
- MERGE
- COPY
- COPY_PARTITIONS_TO_TABLE
- MOVE_PARTITIONS_TO_TABLE
- SWAP_PARTITIONS_BETWEEN_TABLES

Alternatively, rather than automatic enforcement, you can use ANALYZE_CONSTRAINTS to validate primary and unique key constraints after issuing these statements. For more information on enabling and disabling primary key constraints, refer to [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Foreign Key Constraints

A foreign key (FK) is a column that is used to join a table to other tables to ensure referential integrity of the data. A foreign key constraint requires that a column contain only values from the primary key column on a specific dimension table.

You can create a foreign key constraint in the [CREATE TABLE](#) statement, or you can define a foreign key constraint using [ALTER TABLE](#).

A column with a foreign key constraint can contain NULL values if it does not also have a [not NULL](#) constraint, even though the NULL value does not appear in the dimension table's primary key column. This allows rows to be inserted into the table even if the foreign key is not yet known.

You can add a foreign key constraint by referencing the table that contains the primary key. The columns in the referenced table do not need to be specified explicitly.

Examples

Create a table called `inventory` to store inventory data:

```
CREATE TABLE inventory (  
  date_key INTEGER NOT NULL,  
  product_key INTEGER NOT NULL,  
  warehouse_key INTEGER NOT NULL,  
  ...  
);
```

Create a table called `warehouse` to store warehouse information:

```
CREATE TABLE warehouse (  
  warehouse_key INTEGER NOT NULL PRIMARY KEY,  
  warehouse_name VARCHAR(20),  
  ...  
);
```

To ensure referential integrity between the `inventory` and `warehouse` tables, define a foreign key constraint called `fk_inventory_warehouse` on the `inventory` table that references the `warehouse` table:

```
ALTER TABLE inventory  
  ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY(warehouse_key)  
  REFERENCES warehouse(warehouse_key);
```

In this example, the `inventory` table is the *referencing* table and the `warehouse` table is the *referenced* table.

You can also create the foreign key constraint in the CREATE TABLE statement that creates the inventory table, eliminating the need for the ALTER TABLE statement. If you do not specify one or more columns, the PRIMARY KEY of the referenced table is used:

```
CREATE TABLE inventory (  
  date_key INTEGER NOT NULL,  
  product_key INTEGER NOT NULL,  
  warehouse_key INTEGER NOT NULL REFERENCES warehouse(warehouse_key),  
  ...  
);
```

A foreign key can also constrain and reference multiple columns. The following example uses CREATE TABLE to add a foreign key constraint to a pair of columns:

```
CREATE TABLE t1 ( c1 INTEGER PRIMARY KEY,  
  c2 INTEGER,  
  c3 INTEGER,  
  FOREIGN KEY (c2, c3) REFERENCES other_table (c1, c2)  
);
```

The following two examples use ALTER TABLE to add a foreign key constraint to a pair of columns. When you use the CONSTRAINT keyword, you must specify a constraint name:

```
ALTER TABLE t ADD FOREIGN KEY (a, b) REFERENCES other_table(c, d);  
ALTER TABLE t ADD CONSTRAINT fk_cname FOREIGN KEY (a, b) REFERENCES other_table(c, d);
```

Note: The FOREIGN KEY keywords are valid only after the column definition, not *on* the column definition.

Unique Constraints

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all rows in the table.

How to Verify Unique Constraints

Vertica allows you to add a (non-enabled) unique constraint to a column. You can then insert data into that column, regardless of whether that constraint is not unique with respect to other values in that column. If your data does not conform to the declared non-enabled constraints, your queries could yield unexpected results.

You can use [ANALYZE_CONSTRAINTS](#) to check for constraint violations, or you can enable automatic enforcement of unique key constraints. For more information on enabling and disabling unique key constraints, refer to [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Add Unique Column Constraints

There are several ways to add a unique constraint on a column. If you use the `CONSTRAINT` keyword, you must specify a constraint name. The following example adds a `UNIQUE` constraint on the `product_key` column and names it `product_key_UK`:

```
CREATE TABLE product (product_key INTEGER NOT NULL CONSTRAINT product_key_UK UNIQUE,  
    ...  
);
```

Vertica recommends naming constraints, but it is optional:

```
CREATE TABLE product (product_key INTEGER NOT NULL UNIQUE,  
    ...  
);
```

You can specify the constraint after the column definition, with and without naming it:

```
CREATE TABLE product (product_key INTEGER NOT NULL,  
    ...,  
    CONSTRAINT product_key_uk UNIQUE (product_key)  
);  
CREATE TABLE product (  
    product_key INTEGER NOT NULL,  
    ...,  
    UNIQUE (product_key)  
);
```

You can also use `ALTER TABLE` to specify a unique constraint. This example names the constraint `product_key_UK`:

```
ALTER TABLE product ADD CONSTRAINT product_key_UK UNIQUE (product_key);
```

You can use `CREATE TABLE` and `ALTER TABLE` to specify unique constraints on multiple columns. If a unique constraint refers to a group of columns, separate the column names using commas. The column listing specifies that the combination of values in the indicated columns is unique across the whole table, though any one of the columns need not be (and ordinarily isn't) unique:

```
CREATE TABLE dim1 (c1 INTEGER,  
    c2 INTEGER,  
    c3 INTEGER,  
    UNIQUE (c1, c2)  
);
```

Check Constraints

A *check constraint* specifies a SQL predicate (Boolean expression) that is evaluated independently on each row of a table. The predicate cannot access data stored in other tables or database objects, such as sequences. It also cannot access data in other rows of the current table.

As with other constraints, check constraints help ensure data integrity. By default, Vertica automatically enforces check constraints. You have the option of turning this feature off for specific constraints when you create or alter them. You can also set the configuration parameter `EnableNewCheckConstraintsByDefault` to 0 (disabled), which automatically disables all check constraints that you subsequently create or alter. You can override this parameter by explicitly enabling a specific constraint.

For more information on enabling and disabling constraints, refer to [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

The following examples show how you can use check constraints:

Add a constraint named `smp1check` to the column `n1` without specifically enabling or disabling the constraint:

```
=> CREATE TABLE checksample (n1 int CONSTRAINT smp1check CHECK(n1<50),n2 int, n3 int);
```

Add a constraint in the disabled state:

```
=> CREATE TABLE checksample2 (n1 int CONSTRAINT smp1check2 CHECK(n1<50) DISABLED,n2 int, n3 int);
```

Use `ALTER TABLE` to add a constraint to an existing table:

```
=> ALTER TABLE checksample ADD CONSTRAINT smp1check2 CHECK(n2<100);
```

Use `ALTER TABLE` to disable an existing constraint:

```
=> ALTER TABLE checksample ALTER CONSTRAINT smp1check2 DISABLED;
```

When you enable automatic enforcement of check constraints, Vertica applies enforcement for:

- INSERT
- UPDATE
- MERGE

- COPY
- COPY_PARTITIONS_TO_TABLE (Vertica enforces check constraints on the target table.)
- MOVE_PARTITIONS_TO_TABLE (Vertica enforces check constraints on the target table.)
- SWAP_PARTITIONS_BETWEEN_TABLES (Vertica enforces check constraints on both tables.)

Alternatively, rather than automatic enforcement, you can use ANALYZE_CONSTRAINTS to validate constraints after issuing these statements. ANALYZE_CONSTRAINTS reports all constraint violations. For more information on enabling and disabling check constraints, refer to [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#)

Allowed Syntax for Check Constraint Predicates

A check constraint predicate can include:

- Arithmetic and concatenated string operators
- Logical operators — Such as AND, OR, NOT
- WHERE prepositions — Such as CASE, IN, LIKE, BETWEEN, IS (NOT) NULL
- Calls to certain types of functions — Immutable SQL macros, immutable build-in SQL functions (for example, length()) and user-defined scalar functions that are marked as immutable in the component factory.

The following examples show you how you can define hypothetical check constraints on a table. The examples assume that the table contains the referenced attributes (column names), defined as appropriate types. (The last one is an unnamed check constraint.)

```
CONSTRAINT chk_pos_quant CHECK (quantity > 0)
CONSTRAINT chk_pqe CHECK (price*quantity = extended_price)
CONSTRAINT size_sml CHECK (size in ('small', 'medium', 'large', 'x- large'))
CHECK (regexp_like(dept_name, '^[a-z]+$', 'i') OR (dept_name = 'inside sales'))
```

Restrictions on Check Constraints

A check constraint expression must evaluate to a Boolean value. However, Vertica does not support implicit conversion to a Boolean value. For example, Vertica would return an error for the following (invalid) check constraint:

```
CHECK (1) -- produces an error  
CHECK ('hello') -- produces an error
```

You must always enclose a check constraint in parentheses:

```
check (quantity > 0)
```

A check constraint expression cannot include any of the following elements:

- Subqueries — `CHECK (dept_id in (SELECT id FROM dept))`
- Aggregates — `CHECK (quantity < sum(quantity)/2)`
- Window functions — `CHECK (RANK() over () < 3)`
- SQL meta-functions — `CHECK (START_REFRESH('') = 0)`
- References to the epoch column
- References to other tables or objects (for example, sequences), or system context
- Invocation of functions that are not immutable in time and space

Check Constraints and Nulls

If a check constraint expression evaluates to an unknown for a given row because a column within the expression contains a null, the row passes the constraint condition. Vertica evaluates the predicate and considers it satisfied if it resolves to either true or unknown. For example, `check (quantity > 0)` passes validation if `quantity` is null. This result differs from how a `WHERE` clause would work. With a `WHERE` clause, the row would not be included in the result set.

You can prohibit nulls in a check constraint by explicitly including a null check in the check constraint expression. For example, `CHECK (quantity IS NOT NULL AND (quantity > 0))`. You could alternatively include a not null constraint separate from the check constraint.

Check Constraints and SQL Macros

A check constraint can call a SQL macro (a function written in SQL) if the macro is immutable. An *immutable macro* always returns the same value for a given set of arguments.

When you include a macro, Vertica determines if it is immutable. If it is not, Vertica rolls back the DDL statement.

This example shows you can create a macro name, `mycompute`, and then use it within a check constraint expression:

```
=> CREATE OR REPLACE FUNCTION mycompute(j int, name1 varchar)
RETURN int AS BEGIN RETURN (j + length(name1)); END;

=> ALTER TABLE sampletable
ADD CONSTRAINT chk_compute
CHECK(mycompute(weekly_hours, name1)<50);
```

Check Constraints and User-Defined Extensions (UDxs)

A check constraint can call user-defined scalar function (UDSFs), but not other kinds of UDxs. To use a UDSF, you must first mark it as immutable in the UDx factory.

If you use a UDSF within a check constraint, you must verify that the immutable tag on the referenced functions is correct and that the constraint handles null values properly. Otherwise, the check constraint may not work as you intended. In addition, Vertica evaluates the predicate of an enabled check constraint on every row that is loaded or updated. For performance reasons, you may want to avoid invoking a computationally expensive check constraint.

To view a sample UDSF that you can use within a check constraint, refer to [C++ Example: Calling a UDSF from a Check Constraint](#).

Dropping a Check Constraint

Use `ALTER TABLE` with the `DROP CONSTRAINT` option to explicitly drop a constraint.

If you drop a table that includes check constraints, Vertica automatically deletes all check constraints associated with the table. The `CASCADE` option is not needed.

If you drop a table column (or function object):

- If you do not specify the `CASCADE` option, Vertica returns an error if any check constraint references the column. Vertica returns the error even if the constraint is disabled.
- If you specify `CASCADE`, Vertica drops check constraints (both enabled and disabled) along with the table column (or function object).

If you rename a table column, Vertica updates the constraint predicate to use the new name.

Not NULL Constraints

A not NULL constraint specifies that a column *cannot* contain a null value. This means that new rows cannot be inserted or updated unless you specify a value for this column.

You can apply the not NULL constraint when you create a column in a new table, and when you add a column to an existing table ([ALTER TABLE . .ADD COLUMN](#)). You can also add or drop the not NULL constraint on an existing column:

- ALTER TABLE *t* ALTER COLUMN *x* SET NOT NULL
- ALTER TABLE *t* ALTER COLUMN *x* DROP NOT NULL

Important: Using the [SET | DROP] NOT NULL clause does not validate whether column data conforms to the NOT NULL constraint. Use [ANALYZE_CONSTRAINTS](#) to check for constraint violations in a table.

The not NULL constraint is implicitly applied to a column when you add the PRIMARY KEY (PK) constraint. When you designate a column as a primary key, you do not need to specify the not NULL constraint.

However, if you remove the primary key constraint, the not NULL constraint still applies to the column. Use the ALTER COLUMN . .DROP NOT NULL clause of the [ALTER TABLE](#) statement to drop the not NULL constraint after dropping the primary key constraint.

The following statement enforces a not NULL constraint on the `customer_key` column, specifying that the column cannot accept NULL values.

```
CREATE TABLE customer (customer_key INTEGER NOT NULL,  
    ...  
);
```

Dropping Constraints

To drop named constraints, use the [ALTER TABLE](#) command.

The following example drops the constraint fact2fk:

```
=> ALTER TABLE fact2 DROP CONSTRAINT fact2fk;
```

To drop constraints that you did not assign a name to, query the system table [TABLE_CONSTRAINTS](#), which returns both system-generated and user-named constraint names. For example:

```
=> SELECT * FROM TABLE_CONSTRAINTS;
```

If you do not specify a constraint name, Vertica assigns a constraint name that is unique to that table. In the following output, note the system-generated constraint name C_PRIMARY and the user-defined constraint name fk_inventory_date:

```
-[ RECORD 1 ]-----+-----  
constraint_id      | 45035996273707984  
constraint_name    | C_PRIMARY  
constraint_schema_id | 45035996273704966  
constraint_key_count | 1  
foreign_key_count  | 0  
table_id          | 45035996273707982  
foreign_table_id   | 0  
constraint_type    | p  
-[ ... ]-----+-----  
-[ RECORD 9 ]-----+-----  
constraint_id      | 45035996273708016  
constraint_name    | fk_inventory_date  
constraint_schema_id | 0  
constraint_key_count | 1  
foreign_key_count  | 1  
table_id          | 45035996273708014  
foreign_table_id   | 45035996273707994  
constraint_type    | f
```

Once you know the name of the constraint, you can then drop it using the [ALTER TABLE](#) command. (If you do not know the table name, use `table_id` to retrieve `table_name` from the [ALL_TABLES](#) table.)

Remove NOT NULL Constraints

When a column is a primary key and you drop the primary key constraint, the column retains the NOT NULL constraint. To specify that the column now can contain NULL values, use `[DROP`

NOT NULL] to remove the NOT NULL constraint.

Remove (Drop) a NOT NULL constraint on the column using [DROP NOT NULL]:

```
ALTER TABLE T1 ALTER COLUMN x DROP NOT NULL;
```

Important: Using the [SET | DROP] NOT NULL clause does not validate whether the column data conforms to the NOT NULL constraint. Use [ANALYZE_CONSTRAINTS](#) to check for constraint violations in a table.

Limitations of Dropping Constraints

- You cannot drop a primary key constraint if another table has a foreign key constraint that references the primary key.
- If you drop a primary or foreign key constraint, the system does not automatically drop the not NULL constraint on a column. You need to manually drop this constraint if you no longer want it.
- If you drop an enabled primary or unique key constraint, the system drops the associated projection if one was automatically created.

See Also

[ALTER TABLE](#)

Enforcing Primary Key and Foreign Key Constraints

Enforcing (Non-Enabled) Primary Key Constraints

Unless you enable enforcement of primary key constraints, Vertica does not enforce the uniqueness of primary key values when they are loaded into a table. Thus, a key enforcement error can occur unless one dimension row uniquely matches each foreign key value when the table is joined to a dimension table during a query:

Note: Consider using sequences or auto-incrementing columns for primary key columns, which guarantees uniqueness and avoids the constraint enforcement problem and associated overhead. For more information, see [Using Sequences](#).

For information on automatic enforcement of primary key constraints during DML, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Foreign Key Constraint Violations

A table's foreign key constraints are not enforced during data load. Thus, it is possible to load data that causes a constraint violation. Subsequently, a constraint violation error can occur when:

- An inner join query is processed.
- An outer join is treated as an inner join due to the presence of foreign key.

Detecting Constraint Violations Before You Commit Data

To detect constraint violations, you can load data without committing it using the [COPY](#) statement with the NO COMMIT option, and then perform a post-load check using the [ANALYZE_CONSTRAINTS](#) function. If constraint violations exist, you can roll back the load because you have not committed it. For more details, see [Detecting Constraint Violations with ANALYZE_CONSTRAINTS](#).

You can also configure your system to automatically enforce primary key, unique key, and check constraints during DML. For information on automatic enforcement, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Enforcing Primary Key, Unique Key, and Check Constraints Automatically

When you create a new constraint with [CREATE TABLE](#) or [ALTER TABLE](#), you can specify whether the constraint will be automatically enforced. You can also alter a constraint with [ALTER TABLE](#) (using the `ALTER CONSTRAINT` parameter) and specify whether it will be automatically enforced. You enable or disable individual constraints specifically using the `ENABLED` or `DISABLED` options.

In addition, you can create multi-column constraints with [CREATE TABLE](#) or [ALTER TABLE](#). All primary key and unique key constraints are defined at the table level. Check constraints are defined at the column or table level.

By checking any system table with an `is_enabled` column, you can confirm whether a primary key, unique key, or check constraint is currently enabled. The system tables that include an `is_enabled` column are [CONSTRAINT_COLUMNS](#), [TABLE_CONSTRAINTS](#), and [PRIMARY_KEYS](#).

Automatic enforcement applies to current table content and content you later add to the table.

- **Enabling a Constraint on an Empty Table** — If you create an enabled constraint on an empty table, the constraint is enforced on any content you later add to that table.
- **Enabling a Constraint on a Populated Table** — If you use [ALTER TABLE](#) to either enable an existing constraint or add a new constraint that is enabled, the constraint is immediately enforced for the current content, and is enforced for content you subsequently add to the table.

Important: If validation of the current content fails, Vertica completely rolls back the [ALTER TABLE](#) DDL statement that caused the failure.

If you do not specify the `ENABLED` or `DISABLED` option when you create a constraint, the system relies on the setting of the configuration parameter for the respective constraint:

- **EnableNewPrimaryKeysByDefault** — If you specifically create a new primary key constraint but do not enable or disable it, the system relies on the value of the parameter

EnableNewPrimaryKeysByDefault. If the parameter is set to 1 (enabled), the constraint you created is automatically enforced even though you did not specifically enable it when you created it.

- **EnableNewUniqueKeysByDefault** — If you specifically create a new unique key constraint but do not enable or disable it, the system relies on the value of the parameter `EnableNewUniqueKeysByDefault`.
- **EnableNewCheckConstraintsByDefault** — If you specifically create a new check constraint but do not enable or disable it, the system relies on the value of the parameter `EnableNewCheckConstraintsByDefault`. This parameter is the only one of the three that is set to 1 (enabled) by default.

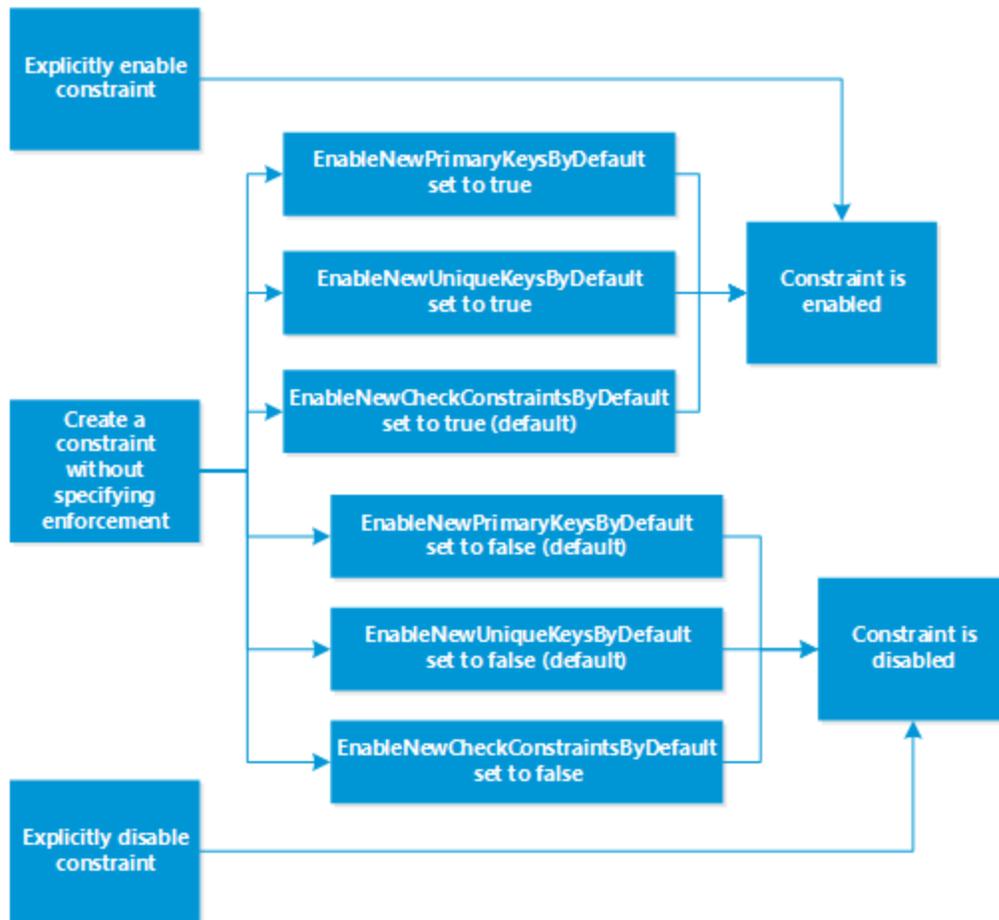
In regards to constraint enablement, you can:

- **Enable or Disable a Constraint When Creating** — You can specifically enable or disable when you create the constraint using `CREATE TABLE` or `ALTER TABLE`. If you do so, the constraint remains enabled or disabled regardless of the setting of the parameters `EnableNewPrimaryKeysByDefault`, `EnableNewUniqueKeysByDefault`, or `EnableNewCheckConstraintsByDefault`.
- **Create a Constraint Without Enabling or Disabling** — You can also create a constraint using `CREATE TABLE` or add a constraint using `ALTER TABLE` without specifically enabling or disabling it using the `ENABLED` or `DISABLED` keyword. If you do so, Vertica looks at the setting of the parameters at the moment you create or alter the constraint to determine whether that constraint is enabled or disabled. Note that, if you alter an existing constraint using `ALTER TABLE ALTER CONSTRAINT`, you must specifically enable or disable it using either the `ENABLED` or `DISABLED` keyword.

Important: When creating a constraint without enabling it, Vertica uses the settings of `EnableNewPrimaryKeysByDefault`, `EnableNewUniqueKeysByDefault`, and `EnableNewCheckConstraintsByDefault` that are in effect at the time of creation.

The following figure summarizes constraint enablement.

Enforcement of Primary, Unique Key, and Check Constraints



Enabling or Disabling Automatic Enforcement of Individual Constraints

To enable or disable individual constraints, use the [CREATE TABLE](#) or [ALTER TABLE](#) statement with the `ENABLED` or `DISABLED` options, as shown in the following examples.

The following sample uses `ALTER TABLE` to create and enable a primary key constraint on a sample table called `mytable`.

```
ALTER TABLE mytable ADD CONSTRAINT primarysample PRIMARY KEY(id) ENABLED;
```

The following sample specifically disables the constraint.

```
ALTER TABLE mytable ALTER CONSTRAINT primarysample DISABLED;
```

The following sample uses CREATE TABLE to create a primary key constraint without explicitly enabling it. In this case, the constraint is enabled only if EnableNewPrimaryKeysByDefault is also enabled. If EnableNewPrimaryKeysByDefault is set to 1 (enabled), then this constraint is enforced. If EnableNewPrimaryKeysByDefault is at its default setting (disabled), then this constraint is not enforced.

```
CREATE TABLE mytable (id INT PRIMARY KEY);
```

The following sample uses CREATE TABLE to create a primary key constraint and enable it. This statement enables the constraint regardless of how you set the parameter EnableNewPrimaryKeysByDefault.

```
CREATE TABLE mytable (id INT PRIMARY KEY ENABLED);
```

The following example uses CREATE TABLE to create and specifically disable the check constraint, chyear. This statement disables the constraint regardless of the setting of how you set the parameter EnableNewCheckConstraintsByDefault.

```
CREATE TABLE emphire (hire_date INT, termination_date INT CONSTRAINT chyear  
CHECK (termination_date >= hire_date) DISABLED, firstnameof VARCHAR, lastnameof VARCHAR);
```

Checking Whether Constraints Are Enabled

Use the SELECT statement to list constraints and confirm whether they are enabled or disabled.

This example shows a query that lists all tables along with their associated primary key, unique, and check constraint types. The query also indicates whether the constraints are enabled or disabled.

```
select table_name, constraint_name, constraint_type, is_enabled  
from v_catalog.constraint_columns where constraint_type in ('p',  
'u', 'c') order by table_name;
```

The following output shows the results of this query. The constraint_type column indicates whether the constraint is a primary key, unique key, or check constraint (p, u, or c, respectively). The is_enabled column indicates whether the constraint is enabled or disabled (t or f respectively).

```
table_name | constraint_name | constraint_type | is_enabled  
-----+-----+-----+-----  
table01   | pksample       | p               | t
```

```
table02    | uniquesample | u          | f  
table03    | checksample  | c          | t  
  
(3 rows)
```

The following example is similar but shows how you can create a query that lists associated columns instead of tables. You could add both tables and columns to the same query, if you want.

```
select column_name, constraint_name, constraint_type, is_enabled  
from v_catalog.constraint_columns where constraint_type in ('p',  
'u', 'c') order by column_name;
```

Sample output follows.

```
column_name | constraint_name | constraint_type | is_enabled  
-----+-----+-----+-----  
col1_key    | pksample       | p              | t  
vendor_key  | uniquesample   | u              | f  
zip_key     | checksample    | c              | t  
  
(2 rows)
```

The following example statement shows how to create a sample table with a multi-column constraint.

```
CREATE TABLE table09 (column1 int, column2 int, CONSTRAINT  
multicsample PRIMARY KEY (column1, column2) ENABLED);
```

Here's the output listing associated columns.

```
column_name | constraint_name | constraint_type | is_enabled  
-----+-----+-----+-----  
column1     | multicsample    | p              | t  
column2     | multicsample    | p              | t  
  
(2 rows)
```

Choosing Default Enforcement for Newly Declared or Modified Constraints

The `EnableNewPrimaryKeysByDefault` and `EnableNewUniqueKeysByDefault` parameter settings govern automatic enforcement of primary key and unique key constraints. The parameter

`EnableNewCheckConstraintsByDefault` governs automatic enforcement of check constraints, and is the only one of the three parameters that is enabled by default.

Important: If you disable enforcement with a parameter, the constraints you create or modify are not enforced unless you specifically enable them using `CREATE TABLE` or `ALTER TABLE`.

You do not need to restart your database once you have set these parameters.

- To enable or disable enforcement of newly created primary keys, set the parameter `EnableNewPrimaryKeysByDefault`. To disable, keep the default setting of 0. To enforce the constraints, set `EnableNewPrimaryKeysByDefault` to 1 to enable.

```
ALTER DATABASE VMart SET EnableNewPrimaryKeysByDefault = 1;
```

- To enable or disable enforcement of newly created constraints for unique keys, set the parameter `EnableNewUniqueKeysByDefault`. To disable, keep the default setting of 0. Set `EnableNewUniqueKeysByDefault` to 1 to enable.

```
ALTER DATABASE VMart SET EnableNewUniqueKeysByDefault = 1;
```

- To enable or disable enforcement of newly created check constraints, set the parameter `EnableNewCheckConstraintsByDefault`. To enable, keep the default setting of 1. Set `EnableNewCheckConstraintsByDefault` to 0 to disable.

```
ALTER DATABASE VMart SET EnableNewCheckConstraintsByDefault = 0;
```

You can check individual parameter values using `SHOW CURRENT`:

```
=> show current EnableNewCheckConstraintsByDefault;
 level | name | setting
-----+-----+-----
DATABASE | EnableNewCheckConstraintsByDefault | 0
```

When you upgrade to Vertica 7.0.x, the primary and unique key constraints in any tables you carry over are disabled. Existing constraints are not automatically enforced. To enable existing constraints and make them automatically enforceable, manually enable each constraint using the `ALTER TABLE ALTER CONSTRAINT` statement. This statement triggers constraint enforcement for the existing table contents. Statements roll back if one or more violations occur.

How Enabled Primary and Unique Key Constraints Affect Locks

If you enable automatic constraint enforcement, Vertica uses an Insert-Validate (IV) lock. The IV lock is needed for operations where the system performs constraint validation for enabled PRIMARY or UNIQUE key constraints. Such operations can include INSERT, COPY, MERGE, UPDATE, MOVE_PARTITIONS_TO_TABLE.

How DML Operates with Constraints

With enforced primary or unique key constraints, DML operates in two-stages.

- **First Stage.** The first stage is the same as it would be for an unenforced constraint, taking, for example, an I lock. (This could be done by several sessions concurrently loading the same table.)
- **Second Stage.** The second stage includes the IV lock. Vertica takes an IV lock to make sure that the data does not violate your constraint. After performing this check, Vertica can commit the data.

Delays in Bulk Loading Caused by Constraint Validation

In bulk load situations, some transactions could be temporarily blocked while primary or unique key constraints are validated. For example:

You could have three sessions (for example, sessions 1, 2 and 3). Each session concurrently has an I lock for a bulk load. Session 1 takes an IV lock to validate constraints. Only one session can hold an IV lock on a given table; other sessions can continue loading the table while holding I locks.

Sessions 2 and 3 wait for session 1 to validate constraints, and then commit, releasing the IV lock. (If session 1 fails, the statement rolls back, and the next session can obtain the IV lock. While sessions can load the table in parallel, an IV lock requires that sessions takes turns obtaining the IV lock for the final stage of constraint validation.)

For information on lock modes and compatibility and conversion matrices, see [Lock Modes](#) in Vertica Concepts. See also the [LOCKS](#) and [LOCK_USAGE](#) sections in the SQL Reference Manual.

Projections for Enabled Primary and Unique Key Constraints

To enforce primary and unique key constraints, Vertica creates special key projections as needed in response to DML or DDL, which are checked for constraint violations. If a constraint violation occurs, Vertica rolls back the statement and any special key projection it created. The system returns an error specifying the unique or primary key constraint that was violated.

If you have added a constraint on a table that is empty, Vertica does not immediately create a special key projection for that constraint. Vertica defers creation of a special key projection until the first row of data is added to the table using a DML or COPY statement. If you add a constraint to a populated table, Vertica chooses an existing projection for enforcement of the constraint, if possible. If none of the existing projections are sufficient to validate the constraint, Vertica creates a new projection for the enabled constraint.

You can check primary and unique key constraint projections by querying the [PROJECTIONS](#) systems table under the [V_CATALOG Schema](#). Each entry applying to a key constraint projection include the column name `IS_KEY_CONSTRAINT_PROJECTION`.

If you drop an enabled primary or unique key constraint, the system may drop an associated projection if one was automatically created. You can drop a specific projection even if a key constraint is enabled:

- If you drop a specific projection without including the `CASCADE` option in your `DROP` statement, Vertica issues a warning about dropping a projection for an enabled constraint.
- If you drop a specific projection and include the `CASCADE` option in your `DROP` statement, Vertica drops the projection without issuing the warning.

In either case, the next time Vertica needs to enforce the constraint for DML, the system creates a new special key projection, unless an existing projection can enforce the same enabled constraint. The time it takes to regenerate a key projection depends upon the volume of the table.

Note: If you subsequently use `ANALYZE_CONSTRAINTS` on a table that has enabled primary or unique key constraints (and thus their associated projections), `ANALYZE_CONSTRAINTS` can leverage the projections previously created for enforcement, resulting in a performance improvement for [ANALYZE_CONSTRAINTS](#).

Deciding Whether to Enable Primary Key, Unique Key, and Check Constraints

You have the option to choose automatic enforcement of primary key, unique key, and check constraints. Depending upon your specific scenario, you can either enable this feature, or use `ANALYZE_CONSTRAINTS` to validate constraints. Consider these factors:

- [Benefits of Enabling Primary Key, Unique Key, and Check Constraints](#)
- [Considerations Before Enabling Constraints](#)
- [Where Constraints Are Enforced](#)
- [Impact of Floating Point Values in Primary Keys When Using Automatic Enforcement](#)
- [Constraint Enforcement Limitations](#)

For more information on using `ANALYZE_CONSTRAINTS` and how automatic enforcement differs, see the Administrator's Guide section, [Detecting Constraint Violations with `ANALYZE_CONSTRAINTS`](#).

Benefits of Enabling Primary Key, Unique Key, and Check Constraints

When you enable primary key, unique key, or check constraints, Vertica validates data before it is inserted. Because you do not need to validate data using `ANALYZE_CONSTRAINTS` after insertion, query speed improves.

Having enabled key constraints, particularly on primary keys, can help the optimizer produce faster query plans, particularly for joins. When a table has an enabled primary key constraint, the optimizer can assume that it has no rows with duplicate values across the key set.

Vertica automatically creates special purpose projections, if necessary, to enforce enabled key constraints. In some cases Vertica can use an existing projection instead.

Considerations Before Enabling Constraints

Multiple factors affect performance. The enforcement process can slow DML and bulk loading.

If you are doing bulk loads, consider the size of your tables and the number of columns in your keys. You could decide to disable automatic enforcement for fact tables, which tend to be larger, but enable enforcement for dimension tables. For fact tables, you could choose manual constraint validation using [ANALYZE_CONSTRAINTS](#), and avoid the load-time overhead of automatic validation.

When you enable automatic enforcement of primary key, unique key, or check constraints, statement rollbacks occur if validation fails during DML. Vertica completely rolls back the statement causing the failure. When deciding to enable automatic enforcement of constraints, consider the impact of statements rolling back on violations. For example, you issue ten insert statements, none of which have committed. If the sixth statement introduces a duplicate, that statement is rolled back. The other statements that do not introduce duplicates can commit.

Note: Vertica performs primary key, unique key, and check constraint enforcement at the SQL statement level rather than the transaction level. You cannot defer enforcement until transaction commit.

Where Constraints Are Enforced

Automatic enforcement of constraints occurs in:

- INSERT statements — Both in single row insertions, and in an INSERT statement that includes the SELECT parameter.
- Bulk loads — On bulk loads that use the COPY statement.
- UPDATE or MERGE statements — All UPDATE and MERGE statements.
- Meta functions — On COPY_PARTITIONS_TO_TABLE, MOVE_PARTITIONS_TO_TABLE and SWAP_PARTITIONS_BETWEEN_TABLES.
- ALTER TABLE statements — On statements that include either the ADD CONSTRAINT or ALTER CONSTRAINT parameters where you are enabling a constraint and the table has existing data.

Impact of Floating Point Values in Primary Keys When Using Automatic Enforcement

Vertica allows NaN, +Inf, and -Inf values in a FLOAT type column, even if the column is part of a primary key. Because FLOAT types provide imprecise arithmetic, Vertica recommends that you not use columns with floating point values within primary keys.

If you do decide to use a FLOAT type within a primary key, note the following in regards to primary key enforcement. (This behavior is the same regardless of whether you enable an automatic constraint or check constraints manually with `ANALYZE_CONSTRAINTS`.)

- For the purpose of enforcing key constraints, Vertica considers two NaNs, (or two +Inf, or two -Inf) values to be equal.
- If a table has an enabled single column primary key constraint of type FLOAT, only one tuple can have a NaN value for the column. Otherwise, the constraint is violated. This is also true for +Inf and -Inf values. Note that this differs from the IEEE 754 standard, which specifies that multiple NaN values are different from each other.
- A join on a single column that contains FLOAT values fails if the table that includes a primary key contains multiple tuples with two NaNs (or +Inf, or -Inf) values.

For information on floating point type, see [DOUBLE PRECISION \(FLOAT\)](#).

Constraint Enforcement Limitations

You can only enable or disable automatic enforcement for primary key, unique key, and check constraints. Vertica does not support automatic enforcement of foreign keys and referential integrity. You can manually validate foreign key constraints using the meta-function [ANALYZE_CONSTRAINTS](#).

Vertica does not support automatic enforcement of constraints on external tables.

Limitations on Using Automatic Enforcement for Local and Global Temporary Tables

This section includes limitations and related notes on using automatic enforcement of primary and unique key constraints with local and global temporary tables. For general information on temporary tables, see [Creating Temporary Tables](#).

Limitations for Local and Global Temporary Tables

Vertica displays an error message if you add an enabled constraint to a local or global temporary table that contains data. Vertica displays the error because it cannot create

projections for enabled constraints on a temporary table if that table is already populated with data.

Limitations Specific to Global Temporary Tables

You cannot use ALTER TABLE to add a new or enable an existing primary or unique key constraint on a global temporary table. Use CREATE TABLE to enable a constraint on a global temporary table.

You can use ALTER TABLE to add a new or enable an existing primary or unique key constraint on a local temporary table if the local temporary table is empty.

Note: You can use ALTER TABLE to disable an already enabled primary or unique key constraint on a global temporary table.

Reporting Constraint Violations

For enabled constraints, Vertica reports multiple constraint violations before roll back. It reports multiple violations for primary key and unique constraints in all circumstances (for example, when you are loading or moving data). However, Vertica behaves differently for check constraints.

Reporting Primary Key and Unique Constraint Violations

The following example shows multiple duplicates in a file named `sampledatafile.tbl`. The sample table named `pktest` includes an enabled primary key comprised of the column `cost`. Vertica reports the duplicates when you attempt to load the table.

```
VMart=> CREATE TABLE pktest(cost int constraint costcons primary key enabled,destination int,timemorn
int,finish int);
CREATE TABLE

VMart=> copy pktest from '/home/dbadmin/test/sampledatafile.tbl' delimiter ',';

ERROR 6745: Duplicate key values: 'cost=106'
-- violates constraint 'public.pktest.costcons'

DETAIL: Additional violations:
Constraint 'public.pktest.costcons':
duplicate key values: 'cost=110'; 'cost=114'; 'cost=115'; 'cost=116'; 'cost=117';
'cost=118'; 'cost=119'; 'cost=120'; 'cost=121'; 'cost=122'; 'cost=123'; 'cost=124';
'cost=125'; 'cost=135'; 'cost=136'; 'cost=137'; 'cost=138'; 'cost=139'; 'cost=150';
'cost=151'; 'cost=20'; 'cost=200'; 'cost=251'; 'cost=252'; 'cost=255'; 'cost=257';
'cost=258'; 'cost=261'; 'cost=263';
```

```
Note: there were additional errors
```

As shown in the example, Vertica provides more complete information on the first violation, including the name of the constraint that was violated:

```
ERROR 6745: Duplicate key values: 'cost=106'  
-- violates constraint 'public.pktest.costcons'
```

For subsequent violations, it reports abbreviated detail, up to a maximum of 30 violations. If there are more than 30 violations, Vertica adds the note "there were additional errors" at the end of the list of violations:

```
DETAIL: Additional violations:  
Constraint 'public.pktest.costcons':  
duplicate key values: 'cost=110'; 'cost=114'; 'cost=115'; 'cost=116'; 'cost=117';  
'cost=118'; 'cost=119'; 'cost=120'; 'cost=121'; 'cost=122'; 'cost=123'; 'cost=124';  
'cost=125'; 'cost=135'; 'cost=136'; 'cost=137'; 'cost=138'; 'cost=139'; 'cost=150';  
'cost=151'; 'cost=20'; 'cost=200'; 'cost=251'; 'cost=252'; 'cost=255'; 'cost=257';  
'cost=258'; 'cost=261'; 'cost=263';
```

```
Note: there were additional errors
```

Refer to the section [Primary Key Constraints](#) for general information on where Vertica applies enforcement for enabled primary key and unique constraints.

Reporting Multiple Check Constraints

For enabled check constraints, Vertica reports only one constraint violation before roll back with three exceptions:

- You add or change an enabled check constraint on a table that already contains data.
- You move or copy partitions to a table, or when you swap partitions between tables.
- You copy a table using the Vertica function `COPY_TABLE`.

Refer to the section [Check Constraints](#) for general information on where Vertica applies enforcement for enabled check constraints.

This example shows multiple check constraint violations using the Vertica function `COPY_TABLE`.

```
VMart=> create table a(a int);  
CREATE TABLE  
VMart=> create table b(a int check(a>5));  
CREATE TABLE
```

```
VMart=> copy b from stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 6
>> 7
>> 8
>> \.
VMart=> copy a from stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> 1
>> 2
>> 3
>> 4
>> \.
VMart=> select copy_table('a','b');
NOTICE 7636: Validating enabled constraints on table 'public.b'...
ERROR 7231: Check constraint 'public.b.C_CHECK' (b.a > 5) violation in table 'public.b': 'a=1'
DETAIL: Additional violations:
Check constraint 'public.b.C_CHECK':
violations: 'a=2'; 'a=3'; 'a=4'
```

Naming Constraints

You can create and name constraints in different ways, as the following examples show. Micro Focus recommends that you name your constraints. If you do not name your constraints, Vertica assigns a name. The examples show various statement syntaxes for creating and altering named and unnamed constraints.

Naming a Primary Key Constraint

Create a table with a named primary key, where the named constraint is `myconstraint`.

```
=> CREATE TABLE addapk (col1 INT CONSTRAINT myconstraint PRIMARY KEY ENABLED, col2 INT);
CREATE TABLE
```

Query the `PRIMARY_KEYS` table to view the constraint.

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_
name IN ('addapk');
```

```
constraint_name | column_name | constraint_type | is_enabled
-----+-----+-----+-----
```

```
myconstraint | col1 | p | t  
(1 row)
```

Creating a Primary Key Constraint Without Naming

Create a table with a primary key constraint, where you do not name the constraint. In this case, Vertica assigns a name to the constraint.

```
=> CREATE TABLE addapk (col1 INT PRIMARY KEY ENABLED, col2 INT);
```

Query the PRIMARY_KEYS table to view the constraint. Vertica has named the constraint C_PRIMARY.

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');
```

```
constraint_name | column_name | constraint_type | is_enabled  
-----+-----+-----+-----  
C_PRIMARY      | col1       | p              | t  
(1 row)
```

Altering Named and Unnamed Constraints

Alter a table that includes a constraint that you named.

```
=> ALTER TABLE addapk ALTER CONSTRAINT myconstraint DISABLED;
```

Alter a table with a constraint to which Vertica has assigned a name.

```
=> ALTER TABLE addapk ALTER CONSTRAINT C_PRIMARY DISABLED;
```

Alter a table by adding a newly named constraint.

```
=> CREATE TABLE addapk (col1 INT, col2 INT);  
CREATE TABLE  
  
=> ALTER TABLE addapk ADD CONSTRAINT myconstraint PRIMARY KEY (COL1) ENABLED;  
WARNING 2623: Column "col1" definition changed to NOT NULL  
ALTER TABLE
```

Alter a table, and let Vertica name the constraint.

```
=> CREATE TABLE addapk (col1 INT, col2 INT);  
CREATE TABLE  
  
=> ALTER TABLE addapk ADD PRIMARY KEY (COL1) ENABLED;  
WARNING 2623: Column "col1" definition changed to NOT NULL
```

```
ALTER TABLE
```

Creating a Multi-Column Constraint

Create a table with a primary key that comprises three columns.

```
=> CREATE TABLE addapk (col1 INT, col2 INT, col3 INT, col4 INT, col5 INT, PRIMARY KEY  
(col1,col2,col3) ENABLED);
```

Query the PRIMARY_KEYS table to view the multi-column primary key constraint.

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_  
name IN ('addapk');
```

constraint_name	column_name	constraint_type	is_enabled
C_PRIMARY	col1	p	t
C_PRIMARY	col2	p	t
C_PRIMARY	col3	p	t

(3 rows)

Create a table with a primary key that comprises three columns, and name the constraint.

```
=> CREATE TABLE addapk (col1 INT, col2 INT, col3 INT, col4 INT, col5 INT, CONSTRAINT myconstraint  
PRIMARY KEY (col1,col2,col3) ENABLED);  
CREATE TABLE
```

Add a three-column primary key after you create the table.

```
=> CREATE TABLE addapk (col1 INT, col2 INT, col3 INT, col4 INT, col5 INT);  
CREATE TABLE  
  
=> ALTER TABLE addapk ADD CONSTRAINT myconstraint PRIMARY KEY (col1,col2,col3) ENABLED;  
WARNING 2623: Column "col1" definition changed to NOT NULL  
WARNING 2623: Column "col2" definition changed to NOT NULL  
WARNING 2623: Column "col3" definition changed to NOT NULL  
ALTER TABLE
```

Creating Multiple Constraint Types

The following example adds three constraints, including primary key, unique, and check constraints.

```
=> CREATE TABLE addapk (col1 INT CONSTRAINT myPKconstraint PRIMARY KEY ENABLED, col2 INT CONSTRAINT  
myUconstraint UNIQUE ENABLED, col3 INT CONSTRAINT myChconstraint CHECK(col1<col2));  
CREATE TABLE
```

Query the CONSTRAINT_COLUMNS table to view the constraints.

```
=> SELECT column_name, constraint_name, constraint_type, is_enabled FROM constraint_columns WHERE  
constraint_type IN ('c','p','u') AND table_name IN ('addapk');
```

column_name	constraint_name	constraint_type	is_enabled
col1	myCHconstraint	c	t
col1	myPKconstraint	p	t
col2	myUconstraint	u	t
col2	myCHconstraint	c	t

(4 rows)

Detecting Constraint Violations with `ANALYZE_CONSTRAINTS`

Use the [ANALYZE_CONSTRAINTS](#) function to manually validate table constraints.

Ways to Use `ANALYZE_CONSTRAINTS`

You can use `ANALYZE_CONSTRAINTS` instead of (or as a supplement to) automatic enforcement of primary key, unique key, and check constraints. For information on automatic enforcement, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

If you do enable primary key, unique key, and check constraints, note that `ANALYZE_CONSTRAINTS` does not check whether the constraints are disabled or enabled. You can use `ANALYZE_CONSTRAINTS` where:

- Primary key, unique key, and check constraints are disabled.
- Enabled and disabled constraints are mixed.

You can use `ANALYZE_CONSTRAINTS` to validate referential integrity of foreign keys. Vertica does not support automatic enforcement of foreign keys.

How to Use `ANALYZE_CONSTRAINTS` to Detect Violations

The `ANALYZE_CONSTRAINTS` function analyzes and reports on constraint violations within the current schema search path. To check for constraint violations:

- Pass an empty argument to check for violations on all tables within the current schema.
- Pass a single table argument to check for violations on the specified table.
- Pass two arguments, a table name and a column or list of columns, to check for violations in those columns.

See the examples in [ANALYZE_CONSTRAINTS](#) for more information.

How ANALYZE_CONSTRAINTS Differs from Automatic Constraint Enforcement

Use ANALYZE_CONSTRAINTS as a reporting mechanism to find constraint violations. ANALYZE_CONSTRAINTS differs from automatic enforcement in that it does not enforce constraints.

Automatic enforcement of primary key, unique key, and check constraints does enforce constraints.

If you have set your constraints for automatic enforcement, during a bulk load Vertica reports on one row and rolls back the statement that introduced the constraint violation. If you want to see a report for all constraint violations:

- Disable the automatic enforcement of your constraints.
- After the bulk load, run ANALYZE_CONSTRAINTS. ANALYZE_CONSTRAINTS analyzes and reports all rows that violate a constraint, without removing the offending rows.

Impact of Floating Point Values In Primary Keys When Using ANALYZE_CONSTRAINTS

Vertica allows NaN, +Inf, and -Inf values in a FLOAT type column, even if the column is part of a primary key. Because FLOAT types provide imprecise arithmetic, Vertica recommends that you not use columns with floating point values within primary keys.

If you do decide to use a FLOAT type within a primary key, note the following in regards to primary key enforcement. (This behavior is the same regardless of whether you enable an automatic constraint or check constraints manually with ANALYZE_CONSTRAINTS.)

- For the purpose of enforcing key constraints, Vertica considers two NaNs, (or two +Inf, or two -Inf) values to be equal.
- If a table has an enabled single column primary key constraint of type FLOAT, only one tuple can have a NaN value for the column. Otherwise, the constraint is violated. This is also true for +Inf and -Inf values. Note that this differs from the IEEE 754 standard, which specifies that multiple NaN values are different from each other.
- A join on a single column that contains FLOAT values fails if the table that includes a primary key contains multiple tuples with two NaNs (or +Inf, or -Inf) values.

For information on floating point type, see [DOUBLE PRECISION \(FLOAT\)](#).

Examples

If you provide the following inputs, Vertica returns one row, indicating one violation, because the same primary key value (10) was inserted into table t1 twice.

```
=> CREATE TABLE t1(c1 INT);
=> ALTER TABLE t1 ADD CONSTRAINT pk_t1 PRIMARY KEY (c1);
=> CREATE PROJECTION t1_p (c1) AS SELECT *
    FROM t1 UNSEGMENTED ALL NODES;
=> INSERT INTO t1 values (10);
=> INSERT INTO t1 values (10); --Duplicate primary key value

=> \x
Expanded display is on.

=> SELECT ANALYZE_CONSTRAINTS('t1');

-[ RECORD 1 ]-----
Schema Name | public
Table Name  | t1
Column Names | c1
Constraint Name | pk_t1
Constraint Type | PRIMARY
Column Values | ('10')
```

In the preceding query, if you give the second INSERT statement any different value, the result is 0 rows (no violations).

In the following example, create a table that contains three integer columns, one a unique key and one a primary key.

```
=> CREATE TABLE table_1(
  a INTEGER,
  b_UK INTEGER UNIQUE,
  c_PK INTEGER PRIMARY KEY
);
```

Insert some values into table table_1 and commit the changes:

```
=> INSERT INTO table_1 values (1, 1, 1);
=> COMMIT;
```

Run ANALYZE_CONSTRAINTS on table table_1. No constraint violations are reported:

```
=> SELECT ANALYZE_CONSTRAINTS('table_1');
(No rows)
```

Insert duplicate unique and primary key values and run ANALYZE_CONSTRAINTS on table table_1 again. Vertica returns two violations: one against the primary key and one against the unique key:

```
=> INSERT INTO table_1 VALUES (1, 1, 1);
=> COMMIT;
=> SELECT ANALYZE_CONSTRAINTS('table_1');

-[ RECORD 1 ]-----
Schema Name      | public
Table Name       | table_1
Column Names     | b_UK
Constraint Name  | C_UNIQUE
Constraint Type  | UNIQUE
Column Values    | ('1')
-[ RECORD 2 ]-----
Schema Name      | public
Table Name       | table_1
Column Names     | c_PK
Constraint Name  | C_PRIMARY
Constraint Type  | PRIMARY
Column Values    | ('1')
```

The following example shows how you can look for constraint violations on only the unique key in the table `table_1`, qualified with its schema name.

```
=> SELECT ANALYZE_CONSTRAINTS('public.table_1', 'b_UK');

-[ RECORD 1 ]-----
Schema Name      | public
Table Name       | table_1
Column Names     | b_UK
Constraint Name  | C_UNIQUE
Constraint Type  | UNIQUE
Column Values    | ('1')

(1 row)
```

The following example shows that you can specify the same column more than once; `ANALYZE_CONSTRAINTS`, however, returns the violation only once.

```
=> SELECT ANALYZE_CONSTRAINTS('table_1', 'c_PK, C_PK');

-[ RECORD 1 ]-----
Schema Name      | public
Table Name       | table_1
Column Names     | c_PK
Constraint Name  | C_PRIMARY
Constraint Type  | PRIMARY
Column Values    | ('1')
```

The following example creates a new table, `table_2`, and inserts a foreign key and different (character) data types.

```
=> CREATE TABLE table_2 (
  x VARCHAR(3),
  y_PK VARCHAR(4),
  z_FK INTEGER REFERENCES table_1(c_PK));
```

Alter the table to create a multicolumn unique key and multicolumn foreign key and create superprojections:

```
=> ALTER TABLE table_2
  ADD CONSTRAINT table_2_multiuk PRIMARY KEY (x, y_PK);
WARNING 2623: Column "x" definition changed to NOT NULL
WARNING 2623: Column "y_PK" definition changed to NOT NULL
```

The following statement inserts a missing foreign key (0) into table `table_2` and commits the changes.

```
=> INSERT INTO table_2 VALUES ('r1', 'Xpk1', 0);
=> COMMIT;
```

Checking for constraints on the table `table_2` in the public schema detects a foreign key violation:

```
=> SELECT ANALYZE_CONSTRAINTS('public.table_2');

-[ RECORD 1 ]-----
Schema Name   | public
Table Name    | table_2
Column Names  | z_FK
Constraint Name | C_FOREIGN
Constraint Type | FOREIGN
Column Values  | ('0')
```

Now add a duplicate value into the unique key and commit the changes:

```
=> INSERT INTO table_2 VALUES ('r2', 'Xpk1', 1);
=> INSERT INTO table_2 VALUES ('r1', 'Xpk1', 1);
=> COMMIT;
```

Checking for constraint violations on table `table_2` detects the duplicate unique key error:

```
=> SELECT ANALYZE_CONSTRAINTS('table_2');

-[ RECORD 1 ]-----
Schema Name   | public
Table Name    | table_2
Column Names  | z_FK
Constraint Name | C_FOREIGN
Constraint Type | FOREIGN
Column Values  | ('0')
-[ RECORD 2 ]-----
Schema Name   | public
Table Name    | table_2
Column Names  | x, y_PK
Constraint Name | table_2_multiuk
Constraint Type | PRIMARY
Column Values  | ('r1', 'Xpk1')
```

Create a table with multicolumn foreign key and then create the superprojections:

```
=> CREATE TABLE table_3(  
    z_fk1 VARCHAR(3),  
    z_fk2 VARCHAR(4));  
=> ALTER TABLE table_3  
    ADD CONSTRAINT table_3_multifk FOREIGN KEY (z_fk1, z_fk2)  
    REFERENCES table_2(x, y_PK);
```

Insert a foreign key that matches a foreign key in table `table_2` and commit the changes:

```
=> INSERT INTO table_3 VALUES ('r1', 'Xpk1');  
=> COMMIT;
```

Check for constraints on table `table_3`. The query detects no violations:

```
=> SELECT ANALYZE_CONSTRAINTS('table_3');  
(No rows)
```

Add a value that does not match and commit the change:

```
=> INSERT INTO table_3 VALUES ('r1', 'NONE');  
=> COMMIT;
```

Check for constraints on table `table_3`. The query detects a foreign key violation:

```
=> SELECT ANALYZE_CONSTRAINTS('table_3');
```

```
-[ RECORD 1 ]---+-----  
Schema Name   | public  
Table Name    | table_3  
Column Names  | z_fk1, z_fk2  
Constraint Name | table_3_multifk  
Constraint Type | FOREIGN  
Column Values  | ('r1', 'NONE')
```

To analyze all constraints on all tables, issue the following statement.

```
SELECT ANALYZE_CONSTRAINTS('');
```

Clean up your database:

```
=> DROP TABLE table_1 CASCADE;  
=> DROP TABLE table_2 CASCADE;  
=> DROP TABLE table_3 CASCADE;
```

To learn how to remove violating rows, see the [DISABLE_DUPLICATE_KEY_ERROR](#) function.

Fixing Constraint Violations

When Vertica finds duplicate primary key or unique values at run time, use the [DISABLE_DUPLICATE_KEY_ERROR](#) function to suppress error messaging. Queries execute as though no constraints are defined on the schema and the effects are session scoped.

The `DISABLE_DUPLICATE_KEY_ERROR` function is for use only for key constraints that are not automatically enabled.

Caution: When called, `DISABLE_DUPLICATE_KEY_ERROR` suppresses data integrity checking and can lead to incorrect query results. Use this function only after you insert duplicate primary keys into a dimension table in the presence of a pre-join projection. Correct the violations and reenables integrity checking with [REENABLE_DUPLICATE_KEY_ERROR](#).

Reenabling Error Reporting

If you ran [DISABLE_DUPLICATE_KEY_ERROR](#) to suppress error reporting while fixing duplicate key violations, you can get incorrect query results going forward. As soon as you fix the violations, run the [REENABLE_DUPLICATE_KEY_ERROR](#) function to restore the default behavior of error reporting.

The effects of this function are session scoped.

Managing Queries

This section covers the following topics:

- [Query Plans](#): Describes how Vertica creates and uses query plans, which optimize access to information in the Vertica database.
- [Directed Queries](#): Shows how to save query plan information.

Query Plans

A query plan is a sequence of step-like paths that the Vertica cost-based query optimizer uses to execute queries. Vertica can produce different query plans for a given query. For each query plan, the query optimizer evaluates the data to be queried: number of rows, column statistics such as number of distinct values (cardinality), distribution of data across nodes. It also evaluates available resources such as CPUs and network topology, and other environment factors. The query optimizer uses this information to develop several potential plans. It then compares plans and chooses one, generally the plan with the [lowest cost](#).

Viewing Query Plans

You can obtain query plans in two ways:

- The [EXPLAIN](#) statement outputs query plans in various [text formats](#).
- Management Console provides a graphical interface for viewing query plans. For detailed information, see [Managing Queries in MC](#) in Using Management Console.

You can also observe the real-time flow of data through a query plan by querying the system table [QUERY_PLAN_PROFILES](#). For more information, see [Profiling Query Plans](#).

EXPLAIN-Generated Query Plans

EXPLAIN returns the optimizer's query plan for executing a specified query. For example:

```
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT customer_name, customer_state FROM customer_dimension WHERE customer_state IN
('MA','NH') AND customer_gender='Male' ORDER BY customer_name LIMIT 10;

Access Path:
+-SELECT LIMIT 10 [Cost: 365, Rows: 10] (PATH ID: 0)
| Output Only: 10 tuples
| Execute on: Query Initiator
| +---> SORT [TOPK] [Cost: 365, Rows: 544] (PATH ID: 1)
| | Order: customer_dimension.customer_name ASC
| | Output Only: 10 tuples
| | Execute on: Query Initiator
| | +---> STORAGE ACCESS for customer_dimension [Cost: 326, Rows: 544] (PATH ID: 2)
| | | Projection: public.customer_dimension_DBD_1_rep_VMartDesign_node0001
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
| | | Filter: (customer_dimension.customer_gender = 'Male')
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator
```

You can use **EXPLAIN** to evaluate choices that the optimizer makes with respect to a given query. If you think query performance is less than optimal, run it through the Database Designer. For more information, see [Incremental Design](#) and [Reducing Run-time of Queries](#).

Query Plan Cost Estimation

The query optimizer chooses a query plan based on cost estimates. The query optimizer uses information from a number of sources to develop potential plans and determine their relative costs. These include:

- Number of table rows
- Column statistics, including: number of distinct values (cardinality), minimum/maximum values, distribution of values, and disk space usage
- Access path that is likely to require fewest I/O operations, and lowest CPU, memory, and network usage
- Available eligible projections

- Join options: join types (merge versus hash joins), join order
- Query predicates
- Data segmentation across cluster nodes

Many important optimizer decisions rely on statistics, which the query optimizer uses to determine the final plan to execute a query. Therefore, it is important that statistics be up to date. Without reasonably accurate statistics, the optimizer could choose a suboptimal plan, which might affect query performance.

Vertica provides hints about statistics in the query plan. See [Query Plan Statistics](#).

Cost versus Execution Runtime

Although costs correlate to query runtime, they do not provide an estimate of actual runtime. For example, if the optimizer determines that Plan A costs twice as much as Plan B, it is likely that Plan A will require more time to run. However, this cost estimate does not necessarily indicate that Plan A will run twice as long as Plan B.

Also, plan costs for different queries are not directly comparable. For example, if the estimated cost of Plan X for query1 is greater than the cost of Plan Y for query2, it is not necessarily true that Plan X's runtime is greater than Plan Y's runtime.

EXPLAIN Output Options

By default, EXPLAIN output represents the query plan as a hierarchy, where each level (path) represents a single database operation that the optimizer uses to execute a query. EXPLAIN output also appends DOT language source so you can display this output graphically with open source [Graphviz](#) tools.

You can qualify EXPLAIN with three output options:

- **JSON** produces the query plan in JSON format.
- **VERBOSE** increases the amount of detail in the rendered query plan. This option is valid for default and JSON output.
- **LOCAL** (on a multi-node database) shows the local query plans assigned to each node, which together comprise the total (global) query plan. If you omit this option, Vertica shows only the global query plan. Local query plan are shown only in DOT language source, which can be rendered in [Graphviz](#).

You can also [write EXPLAIN output to a file](#).

JSON Output

The following EXPLAIN statement specifies the same query shown earlier, but this time specifies to produce output in JSON format:

```
=> EXPLAIN JSON SELECT customer_name, customer_state FROM customer_dimension WHERE customer_state IN ('MA','NH') AND customer_gender='Male' ORDER BY customer_name LIMIT 10;
```

```
QUERY PLAN DESCRIPTION:
```

```
-----  
Opt Vertica Options
```

```
-----  
PLAN_OUTPUT_JSON
```

```
EXPLAIN JSON SELECT customer_name, customer_state FROM customer_dimension WHERE customer_state IN ('MA','NH') AND customer_gender='Male' ORDER BY customer_name LIMIT 10;
```

```
-----  
JSON format:
```

```
{  
  "PATH_ID" : 0,  
  "PATH_NAME" : "SELECT",  
  "EXTRA" : " LIMIT 10",  
  "COST" : 365.000000,  
  "ROWS" : 10.000000,  
  "COST_STATUS" : "GOOD",  
  "TUPLE_LIMIT" : 10,  
  "EXECUTE_NODE" : "Query Initiator",  
  "INPUT" : {  
    "PATH_ID" : 1,  
    "PATH_NAME" : "SORT",  
    "EXTRA" : "[TOPK]",  
    "COST" : 365.000000,  
    "ROWS" : 544.000000,  
    "COST_STATUS" : "GOOD",  
    "ORDER" : ["customer_dimension.customer_name", "customer_dimension.customer_state"],  
    "TUPLE_LIMIT" : 10,  
    "EXECUTE_NODE" : "Query Initiator",  
    "INPUT" : {  
      "PATH_ID" : 2,  
      "PATH_NAME" : "STORAGE ACCESS",  
      "EXTRA" : "for customer_dimension",  
      "COST" : 326.000000,  
      "ROWS" : 544.000000,  
      "COST_STATUS" : "GOOD",  
      "PROJECTION" : "public.customer_dimension_DBD_1_rep_VMartDesign_node0001",  
      "MATERIALIZER" : ["customer_dimension.customer_state", "customer_dimension.customer_name"],  
      "FILTER" : ["(customer_dimension.customer_gender = 'Male')", "(customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))"],
```

```

        "EXECUTE_NODE" : "Query Initiator"
    }
}
}
End JSON format
-----
(51 rows)

```

VERBOSE Output

The following EXPLAIN statement specifies the same query shown earlier, but this time specifies to produce verbose output (added information is set off in bold):

```

QUERY PLAN DESCRIPTION:
-----

Opt Vertica Options
-----
PLAN_OUTPUT_SUPER_VERBOSE

EXPLAIN VERBOSE SELECT customer_name, customer_state FROM customer_dimension
WHERE customer_state IN ('MA', 'NH') AND customer_gender='Male'
ORDER BY customer_name LIMIT 10;

Access Path:
+-SELECT LIMIT 10 [Cost: 365.000000, Rows: 10.000000 Disk(B): 0.000000 CPU(B): 0.000000 Memory(B):
0.000000 Netwrk(B): 0.000000 Parallelism: 1.000000] [OutRowSz (B): 274] (PATH ID: 0)
| Output Only: 10 tuples
| Execute on: Query Initiator
| Sort Key: (customer_dimension.customer_name)
| LDISTRIB_UNSEGMENTED
| +---> SORT [TOPK] [Cost: 365.000000, Rows: 544.000000 Disk(B): 0.000000 CPU(B): 1275443.584695
Memory(B): 149056.000000 Netwrk(B): 0.000000 Parallelism: 1.000000] [OutRowSz (B): 274] (PATH ID: 1)
| | Order: customer_dimension.customer_name ASC
| | Output Only: 10 tuples
| | Execute on: Query Initiator
| | Sort Key: (customer_dimension.customer_name)
| | LDISTRIB_UNSEGMENTED
| | +---> STORAGE ACCESS for customer_dimension [Cost: 326.000000, Rows: 544.000000 Disk(B): 0.000000
CPU(B): 0.000000 Memory(B): 0.000000 Netwrk(B): 0.000000 Parallelism: 1.000000] [OutRowSz (B): 274]
(PATH ID: 2)
| | | Column Cost Aspects: [ Disk(B): 410928.432432 CPU(B): 111588.324324 Memory(B):
4498570.572304 Netwrk(B): 0.000000 Parallelism: 1.000000 ]
| | | Projection: public.customer_dimension_DBD_1_rep_VMartDesign_node0001
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
| | | Filter: (customer_dimension.customer_gender = 'Male')/* sel=0.351351 ndv= 2 */
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))/* sel=0.030928 ndv=
4 */
| | | Execute on: Query Initiator
| | | Sort Key: (customer_dimension.customer_type, customer_dimension.store_membership_card,
customer_dimension.title, customer_dimension.customer_region, customer_dimension.number_of_children,
customer_dimension.customer_key)

```

Local Output

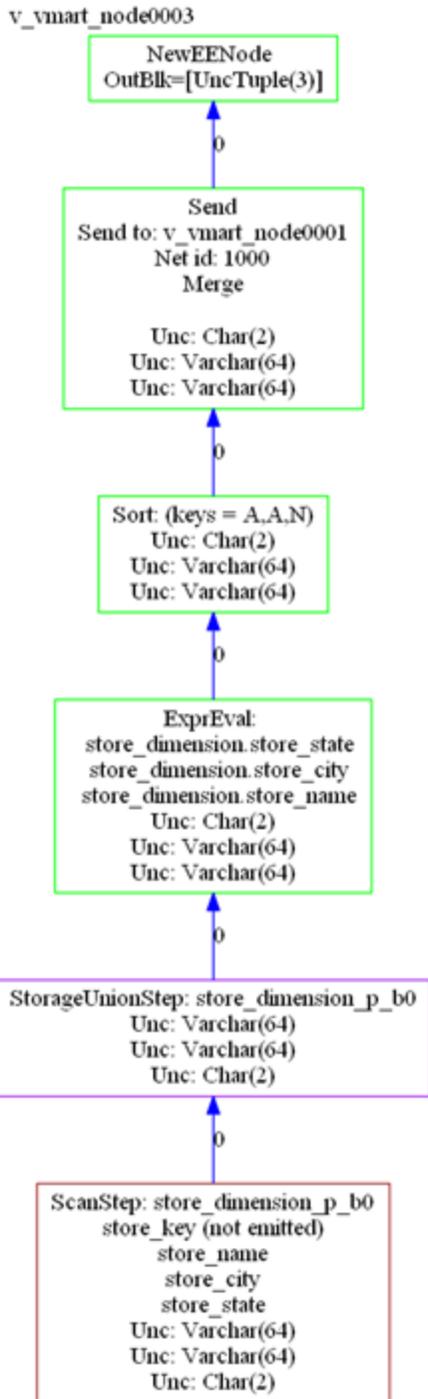
The following statement includes the LOCAL option:

```
=> EXPLAIN LOCAL SELECT store_name, store_city, store_state  
FROM store.store_dimension ORDER BY store_state ASC, store_city ASC;
```

The output includes GraphViz source, which describes the local query plans assigned to each node. For example, output for this statement on a three-node database includes a GraphViz description of the following query plan for one node (v_vmart_node0003):

```
-----  
PLAN: v_vmart_node0003 (GraphViz Format)  
-----  
digraph G {  
graph [rankdir=BT, label = "v_vmart_node0003\n", labelloc=t, labeljust=l ordering=out]  
0[label = "NewEENode \nOutBlk=[UncTuple(3)]", color = "green", shape = "box"];  
1[label = "Send\nSend to: v_vmart_node0001\nNet id: 1000\nMerge\n\nUnc: Char(2)\nUnc: Varchar  
(64)\nUnc: Varchar(64)", color = "green", shape = "box";  
2[label = "Sort: (keys = A,A,N)\nUnc: Char(2)\nUnc: Varchar(64)\nUnc: Varchar(64)", color = "green",  
shape = "box"];  
3[label = "ExprEval: \n store_dimension.store_state\n store_dimension.store_city\n store_  
dimension.store_name\nUnc: Char(2)\nUnc: Varchar(64)\nUnc: Varchar(64)  
", color = "green", shape = "box";  
4[label = "StorageUnionStep: store_dimension_p_b0\nUnc: Varchar(64)\nUnc: Varchar(64)\nUnc: Char  
(2)", color = "purple", shape = "box";  
5[label = "ScanStep: store_dimension_p_b0\nstore_key (not emitted)\nstore_name\nstore_city\nstore_  
state\nUnc: Varchar(64)\nUnc: Varchar(64)\nUnc: Char(2)", color  
= "brown", shape = "box"];  
1->0 [label = "\u0000",color = "blue"];  
2->1 [label = "\u0000",color = "blue"];  
3->2 [label = "\u0000",color = "blue"];  
4->3 [label = "\u0000",color = "blue"];  
5->4 [label = "\u0000",color = "blue"];  
}
```

GraphViz renders this output as follows:



Writing EXPLAIN Output to a File

To save EXPLAIN output to a file, use the vsql \o command:

1. Turn on saving output to a file using the `\o` command. For example:

```
vmartdb=> \o /home/dbadmin/my-plan-output
```

2. Use `EXPLAIN` to obtain the desired query plan:

```
vmartdb=> EXPLAIN SELECT * FROM customer_dimension;
```

3. Turn off saving output to a file using the `\o` command.

```
vmartdb=> \o
```

Note: The `\o` command continues to write command-line output to the file until you reissue the `\o` command.

Query Plan Information and Structure

Depending on the query and database schema, `EXPLAIN` output includes the following information:

- Tables referenced by the statement
- Estimated costs
- Estimated row cardinality
- Path ID, an integer that links to error messages and profiling counters so you troubleshoot performance issues more easily. For more information, see [Profiling Query Plans](#).
- Data operations such as `SORT`, `FILTER`, `LIMIT`, and `GROUP BY`
- Projections used
- Information about statistics—for example, whether they are current or out of range
- Algorithms chosen for operations into the query, such as `HASH/MERGE` or `GROUPBY HASH/GROUPBY PIPELINED`
- Data redistribution (broadcast, segmentation) across cluster nodes

Example

In the EXPLAIN output that follows, the optimizer processes a query in three steps, where each step identified by a unique path ID:

- 0: Limit
- 1: Sort
- 2: Storage access and filter

QUERY PLAN DESCRIPTION:

```
EXPLAIN SELECT customer_name, customer_state FROM customer_dimension  
WHERE customer_state IN ('MA','NH') AND customer_gender = 'Male'  
ORDER BY customer_name LIMIT 10;
```

Access Path:

```
+--SELECT LIMIT 10 [Cost: 365, Rows: 10] (PATH ID: 0) ← Limit  
| Output Only: 10 tuples  
| Execute on: Query Initiator  
| +----> SORT [TOPK] [Cost: 365, Rows: 544] (PATH ID: 1) ← Sort  
| | Order: customer_dimension.customer_name ASC  
| | Output Only: 10 tuples  
| | Execute on: Query Initiator  
| | +----> STORAGE ACCESS for customer_dimension [Cost: 326, Rows: 544] (PATH ID: 2)  
| | | Projection: public.customer_dimension_DBD_1_rep_VMartDesign_node0001  
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name  
| | | Filter: (customer_dimension.customer_gender = 'Male')  
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))  
| | | Execute on: Query Initiator
```

Storage access and filter

Note: A storage access operation can scan more than the columns in the SELECT list— for example, columns referenced in WHERE clause.

Query Plan Statistics

If you query a table whose statistics are unavailable or out-of-date, the optimizer might choose a sub-optimal query plan.

You can resolve many issues related to table statistics by calling [ANALYZE_STATISTICS](#). This function let you update statistics at various scopes: one or more table columns, a single table, or all database tables.

If you update statistics and find that the query still performs sub-optimally, run your query through Database Designer and choose incremental design as the design type.

For detailed information about updating database statistics, see [Collecting Database Statistics](#).

Statistics Hints in Query Plans

Query plans can contain information about table statistics through two hints: `NO STATISTICS` and `STALE STATISTICS`. For example, the following query plan fragment includes `NO STATISTICS` to indicate that histograms are unavailable:

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
```

The following query plan fragment includes `STALE STATISTICS` to indicate that the predicate has fallen outside the histogram range:

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 35, Rows: 1 (STALE STATISTICS)]
```

Cost and Rows Path

The following EXPLAIN output shows the Cost operator:

```
Access Path: +-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)
| Output Only: 10 tuples
| Execute on: Query Initiator
| +---> SORT [Cost: 370, Rows: 544] (PATH ID: 1)
| | Order: customer_dimension.customer_name ASC
| | Output Only: 10 tuples
| | Execute on: Query Initiator
| | +---> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)
| | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
| | | Filter: (customer_dimension.customer_gender = 'Male')
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator
```

The Row operator is the number of rows the optimizer estimates the query will return. Letters after numbers refer to the units of measure (K=thousand, M=million, B=billion, T=trillion), so the output for the following query indicates that the number of rows to return is 50 *thousand*.

```
=> EXPLAIN SELECT customer_gender FROM customer_dimension;
Access Path:
+-STORAGE ACCESS for customer_dimension [Cost: 17, Rows: 50K (3 RLE)] (PATH ID: 1)
| Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| Materialize: customer_dimension.customer_gender
| Execute on: Query Initiator
```

The reference to (3 RLE) in the STORAGE ACCESS path means that the optimizer estimates that the storage access operator returns 50K rows. Because the column is run-length encoded (RLE), the real number of RLE rows returned is only three rows:

- 1 row for female
- 1 row for male
- 1 row that represents unknown (NULL) gender

Note: See [Query Plans](#) for more information about how the optimizer estimates cost.

Projection Path

You can see which projections the optimizer chose for the query plan by looking at the Projection path in the textual output:

```
EXPLAIN SELECT
  customer_name,
  customer_state
FROM customer_dimension
WHERE customer_state in ('MA','NH')
AND customer_gender = 'Male'
ORDER BY customer_name
LIMIT 10;
Access Path:
+-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)
| Output Only: 10 tuples
| Execute on: Query Initiator
| +---> SORT [Cost: 370, Rows: 544] (PATH ID: 1)
| | Order: customer_dimension.customer_name ASC
| | Output Only: 10 tuples
| | Execute on: Query Initiator
| | +---> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)
| | | Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
| | | Filter: (customer_dimension.customer_gender = 'Male')
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator
```

The query optimizer automatically picks the best projections, but without reasonably accurate statistics, the optimizer could choose a suboptimal projection or join order for a query. For details, see [Collecting Statistics](#).

Vertica considers which projection to choose for a plan by considering the following aspects:

- How columns are joined in the query
- How the projections are grouped or sorted
- Whether SQL analytic operations applied
- Any column information from a projection's storage on disk

As Vertica scans the possibilities for each plan, projections with the higher initial costs could end up in the final plan because they make joins cheaper. For example, a query can be answered with many possible plans, which the optimizer considers before choosing one of them. For efficiency, the optimizer uses sophisticated algorithms to prune intermediate partial plan fragments with higher cost. The optimizer knows that intermediate plan fragments might initially look bad (due to high storage access cost) but which produce excellent final plans due to other optimizations that it allows.

If your statistics are up to date but the query still performs poorly, run the query through the Database Designer. For details, see [Incremental Design](#)

Tips

- To test different segmented projections, refer to the projection by name in the query.
- For optimal performance, write queries so the columns are sorted the same way that the projection columns are sorted.

See Also

- [Reducing Query Run Time](#)
- [Creating Custom Designs](#)
- [Physical Schema](#)

Join Path

Just like a join query, which references two or more tables, the Join step in a query plan has two input branches:

- The left input, which is the outer table of the join
- The right input, which is the inner table of the join

In the following query, the T1 table is the left input because it is on the left side of the JOIN keyword, and the T2 table is the right input, because it is on the right side of the JOIN keyword:

```
SELECT * FROM T1 JOIN T2 ON T1.x = T2.x;
```

Outer versus Inner Join

Query performance is better if the small table is used as the inner input to the join. The query optimizer automatically reorders the inputs to joins to ensure that this is the case unless the join in question is an outer join.

Note: If the configuration parameter `EnableForceOuter` is set to 1, you can control join inputs for specific tables through `ALTER TABLE . . FORCE OUTER`. For details, see [Controlling Join Inputs](#) in *Analyzing Data*.

The following example shows a query and its plan for a left outer join:

```
=> EXPLAIN SELECT CD.annual_income,OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> LEFT OUTER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [LeftOuter] [Cost: 4K, Rows: 5M] (PATH ID: 1)
|   Join Cond: (CD.customer_key = OSI.customer_key)
|   Materialize at Output: OSI.sale_date_key
|   Execute on: All Nodes
|   +-- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
|   |   Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
|   |   Materialize: OSI.customer_key
|   |   Execute on: All Nodes
|   +-- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
|   |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
|   |   Materialize: CD.annual_income, CD.customer_key
|   |   Execute on: All Nodes
```

The following example shows a query and its plan for a full outer join:

```
=> EXPLAIN SELECT CD.annual_income,OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> FULL OUTER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [FullOuter] [Cost: 18K, Rows: 5M] (PATH ID: 1) Outer (RESEGMENT) Inner (FILTER)
|   Join Cond: (CD.customer_key = OSI.customer_key)
|   Execute on: All Nodes
|   +-- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
|   |   Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
|   |   Materialize: OSI.sale_date_key, OSI.customer_key
|   |   Execute on: All Nodes
|   +-- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
|   |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
|   |   Materialize: CD.annual_income, CD.customer_key
|   |   Execute on: All Nodes
```

Hash and Merge Joins

Vertica has two join algorithms to choose from: merge join and hash join. The optimizer automatically chooses the most appropriate algorithm, given the query and projections in a

system.

For the following query, the optimizer chooses a hash join.

```
=> EXPLAIN SELECT CD.annual_income,OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [Cost: 4K, Rows: 5M] (PATH ID: 1)
|   Join Cond: (CD.customer_key = OSI.customer_key)
|   Materialize at Output: OSI.sale_date_key
|   Execute on: All Nodes
| +- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
| |   Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
| |   Materialize: OSI.customer_key
| |   Execute on: All Nodes
| +- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
| |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| |   Materialize: CD.annual_income, CD.customer_key
| |   Execute on: All Nodes
```

Tip: If you get a hash join when you are expecting a merge join, it means that at least one of the projections is not sorted on the join column (for example, `customer_key` in the preceding query). To facilitate a merge join, you might need to create different projections that are sorted on the join columns.

In the next example, the optimizer chooses a merge join. The optimizer's first pass performs a merge join because the inputs are presorted, and then it performs a hash join.

```
=> EXPLAIN SELECT count(*) FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key
-> INNER JOIN product_dimension PD ON PD.product_key = OSI.product_key;
Access Path:
+-GROUPBY NOTHING [Cost: 8K, Rows: 1] (PATH ID: 1)
|   Aggregates: count(*)
|   Execute on: All Nodes
| +--> JOIN HASH [Cost: 7K, Rows: 5M] (PATH ID: 2)
| |   Join Cond: (PD.product_key = OSI.product_key)
| |   Materialize at Input: OSI.product_key
| |   Execute on: All Nodes
| | +- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 4K, Rows: 5M] (PATH ID: 3)
| | |   Join Cond: (CD.customer_key = OSI.customer_key)
| | |   Execute on: All Nodes
| | | +- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 4)
| | | |   Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
| | | |   Materialize: OSI.customer_key
| | | |   Execute on: All Nodes
| | | +- Inner -> STORAGE ACCESS for CD [Cost: 132, Rows: 50K] (PATH ID: 5)
| | | |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| | | |   Materialize: CD.customer_key
| | | |   Execute on: All Nodes
| | +- Inner -> STORAGE ACCESS for PD [Cost: 152, Rows: 60K] (PATH ID: 6)
| | |   Projection: public.product_dimension_DBD_2_rep_vmartdb_design_vmartdb_design_node0001
| | |   Materialize: PD.product_key
| | |   Execute on: All Nodes
```

Inequality Joins

Vertica processes joins with equality predicates very efficiently. The query plan shows equality join predicates as join condition (Join Cond).

```
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD
-> ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [Cost: 4K, Rows: 5M] (PATH ID: 1)
| Join Cond: (CD.customer_key = OSI.customer_key)
| Materialize at Output: OSI.sale_date_key
| Execute on: All Nodes
| +-- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
| | Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
| | Materialize: OSI.customer_key
| | Execute on: All Nodes
| +-- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
| | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| | Materialize: CD.annual_income, CD.customer_key
| | Execute on: All Nodes
```

However, inequality joins are treated like cross joins and can run less efficiently, which you can see by the change in cost between the two queries:

```
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD
-> ON CD.customer_key < OSI.customer_key;
Access Path:
+-JOIN HASH [Cost: 98M, Rows: 5M] (PATH ID: 1)
| Join Filter: (CD.customer_key < OSI.customer_key)
| Materialize at Output: CD.annual_income
| Execute on: All Nodes
| +-- Outer -> STORAGE ACCESS for CD [Cost: 132, Rows: 50K] (PATH ID: 2)
| | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| | Materialize: CD.customer_key
| | Execute on: All Nodes
| +-- Inner -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 3)
| | Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
| | Materialize: OSI.sale_date_key, OSI.customer_key
| | Execute on: All Nodes
```

Event Series Joins

Event series joins are denoted by the INTERPOLATED path.

```
=> EXPLAIN SELECT * FROM hTicks h FULL OUTER JOIN aTicks a -> ON (h.time INTERPOLATE PREVIOUS
Access Path:
+-JOIN (INTERPOLATED) [FullOuter] [Cost: 31, Rows: 4 (NO STATISTICS)] (PATH ID: 1)
Outer (SORT ON JOIN KEY) Inner (SORT ON JOIN KEY)
```

```
| Join Cond: (h."time" = a."time")
| Execute on: Query Initiator
| +-- Outer -> STORAGE ACCESS for h [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID: 2)
| | Projection: public.hTicks_node0004
| | Materialize: h.stock, h."time", h.price
| | Execute on: Query Initiator
| +-- Inner -> STORAGE ACCESS for a [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID: 3)
| | Projection: public.aTicks_node0004
| | Materialize: a.stock, a."time", a.price
| | Execute on: Query Initiator
```

Path ID

The `PATH ID` is a unique identifier that Vertica assigns to each operation (path) within a query plan. The same identifier is shared by:

- [Query plans](#)
- Join error messages
- System tables [EXECUTION_ENGINE_PROFILES](#) and [QUERY_PLAN_PROFILES](#)

Path IDs can help you trace issues to their root cause. For example, if a query returns a join error, preface the query with `EXPLAIN` and look for `PATH ID n` in the query plan to see which join in the query had the problem.

For example, the following `EXPLAIN` output shows the path ID for each path in the optimizer's query plan:

```
=> EXPLAIN SELECT * FROM fact JOIN dim ON x=y JOIN ext on y=z;
Access Path:
+-JOIN MERGEJOIN(inputs presorted) [Cost: 815, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Join Cond: (dim.y = ext.z)
| Materialize at Output: fact.x
| Execute on: All Nodes
| +-- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 408, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
| | Join Cond: (fact.x = dim.y)
| | Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
| | | Projection: public.fact_super
| | | Materialize: fact.x
| | | Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for dim [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 4)
| | | Projection: public.dim_super
| | | Materialize: dim.y
| | | Execute on: All Nodes
| +-- Inner -> STORAGE ACCESS for ext [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 5)
| | Projection: public.ext_super
| | Materialize: ext.z
| | Execute on: All Nodes
```

Filter Path

The **Filter** step evaluates predicates on a single table. It accepts a set of rows, eliminates some of them (based on the criteria you provide in your query), and returns the rest. For example, the optimizer can filter local data of a join input that will be joined with another re-segmented join input.

The following statement queries the `customer_dimension` table and uses the `WHERE` clause to filter the results only for male customers in Massachusetts and New Hampshire.

```
EXPLAIN SELECT
  CD.customer_name,
  CD.customer_state,
  AVG(CD.customer_age) AS avg_age,
  COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH') AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name;
```

The query plan output is as follows:

```
Access Path:
+-GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 1)
|  Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
|  Group By: CD.customer_state, CD.customer_name
|  Execute on: Query Initiator
| +---> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 2)
| |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| |   Materialize: CD.customer_state, CD.customer_name, CD.customer_age
| |   Filter: (CD.customer_gender = 'Male')
| |   Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))
| |   Execute on: Query Initiator
```

GROUP BY Paths

A **GROUP BY** operation has two algorithms:

- **GROUPBY PIPELINED** requires that inputs be presorted on the columns specified in the group, which means that Vertica need only retain data in the current group in memory. **GROUPBY PIPELINED** operations are preferred because they are generally faster and require less memory than **GROUPBY HASH**. **GROUPBY PIPELINED** is especially useful for queries that process large numbers of high-cardinality group by columns or **DISTINCT** aggregates.
- **GROUPBY HASH** input is not sorted by the group columns, so Vertica builds a hash table on those group columns in order to process the aggregates and group by expressions.


```
=> EXPLAIN SELECT COUNT(DISTINCT annual_income) FROM customer_dimension
WHERE customer_gender = 'Female';

Access Path: +-GROUPBY NOTHING [Cost: 161, Rows: 1] (PATH ID: 1)
| Aggregates: count(DISTINCT customer_dimension.annual_income)
| +---> GROUPBY PIPELINED [Cost: 158, Rows: 10K] (PATH ID: 2)
| | Group By: customer_dimension.annual_income
| | +---> STORAGE ACCESS for customer_dimension [Cost: 144, Rows: 47K] (PATH ID: 3)
| | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| | | Materialize: customer_dimension.annual_income
| | | Filter: (customer_dimension.customer_gender = 'Female')
```

Tip: If EXPLAIN reports GROUPBY HASH, modify the projection design to force it to use GROUPBY PIPELINED.

Sort Path

The SORT operator sorts the data according to a specified list of columns. The EXPLAIN output indicates the sort expressions and if the sort order is ascending (ASC) or descending (DESC).

For example, the following query plan shows the column list nature of the SORT operator:

```
EXPLAIN SELECT
  CD.customer_name,
  CD.customer_state,
  AVG(CD.customer_age) AS avg_age,
  COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH')
  AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name
ORDER BY avg_age, customer_name;

Access Path:
+-SORT [Cost: 422, Rows: 544] (PATH ID: 1)
| Order: (<SVAR> / float8(<SVAR>)) ASC, CD.customer_name ASC
| Execute on: Query Initiator
| +---> GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 2)
| | Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
| | Group By: CD.customer_state, CD.customer_name
| | Execute on: Query Initiator
| | +---> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 3)
| | | Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
| | | Materialize: CD.customer_state, CD.customer_name, CD.customer_age
| | | Filter: (CD.customer_gender = 'Male')
| | | Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator
```

If you change the sort order to descending, the change appears in the query plan:

```
EXPLAIN SELECT
  CD.customer_name,
  CD.customer_state,
```

```
AVG(CD.customer_age) AS avg_age,  
COUNT(*) AS count  
FROM customer_dimension CD  
WHERE CD.customer_state in ('MA','NH')  
AND CD.customer_gender = 'Male'  
GROUP BY CD.customer_state, CD.customer_name  
ORDER BY avg_age DESC, customer_name;  
Access Path:  
+-SORT [Cost: 422, Rows: 544] (PATH ID: 1)  
| Order: (<SVAR> / float8(<SVAR>)) DESC, CD.customer_name ASC  
| Execute on: Query Initiator  
| +---> GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 2)  
| | Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)  
| | Group By: CD.customer_state, CD.customer_name  
| | Execute on: Query Initiator  
| | +----> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 3)  
| | | Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001  
| | | Materialize: CD.customer_state, CD.customer_name, CD.customer_age  
| | | Filter: (CD.customer_gender = 'Male')  
| | | Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))  
| | | Execute on: Query Initiator
```

Limit Path

The LIMIT path restricts the number of result rows based on the LIMIT clause in the query. Using the LIMIT clause in queries with thousands of rows might increase query performance.

The optimizer pushes the LIMIT operation as far down as possible in queries. A single LIMIT clause in the query can generate multiple Output Only plan annotations.

```
=> EXPLAIN SELECT COUNT(DISTINCT annual_income) FROM customer_dimension LIMIT 10;  
Access Path:  
+-SELECT LIMIT 10 [Cost: 161, Rows: 10] (PATH ID: 0)  
| Output Only: 10 tuples  
| +---> GROUPBY NOTHING [Cost: 161, Rows: 1] (PATH ID: 1)  
| | Aggregates: count(DISTINCT customer_dimension.annual_income)  
| | Output Only: 10 tuples  
| | +----> GROUPBY HASH (SORT OUTPUT) [Cost: 158, Rows: 10K] (PATH ID: 2)  
| | | Group By: customer_dimension.annual_income  
| | | +----> STORAGE ACCESS for customer_dimension [Cost: 132, Rows: 50K] (PATH ID: 3)  
| | | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001  
| | | | Materialize: customer_dimension.annual_income
```

Data Redistribution Path

The optimizer can redistribute join data in two ways:

- Broadcasting
- Resegmentation

Broadcasting

Broadcasting sends a complete copy of an intermediate result to all nodes in the cluster. Broadcast is used for joins in the following cases:

- One table is very small (usually the inner table) compared to the other.
- Vertica can avoid other large upstream resegmentation operations.
- Outer join or subquery semantics require one side of the join to be replicated.

For example:

```
=> EXPLAIN SELECT * FROM T1 LEFT JOIN T2 ON T1.a > T2.y;
Access Path:
+-JOIN HASH [LeftOuter] [Cost: 40K, Rows: 10K (NO STATISTICS)] (PATH ID: 1) Inner (BROADCAST)
|   Join Filter: (T1.a > T2.y)
|   Materialize at Output: T1.b
|   Execute on: All Nodes
|   +-- Outer -> STORAGE ACCESS for T1 [Cost: 151, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
|   |       Projection: public.T1_b0
|   |       Materialize: T1.a
|   |       Execute on: All Nodes
|   +-- Inner -> STORAGE ACCESS for T2 [Cost: 302, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
|   |       Projection: public.T2_b0
|   |       Materialize: T2.x, T2.y
|   |       Execute on: All Nodes
```

Resegmentation

Resegmentation takes an existing projection or intermediate relation and resegments the data evenly across all cluster nodes. At the end of the resegmentation operation, every row from the input relation is on exactly one node. Resegmentation is the operation used most often for distributed joins in Vertica if the data is not already segmented for local joins. For more detail, see [Identical Segmentation](#) in Analyzing Data.

For example:

```
=> CREATE TABLE T1 (a INT, b INT) SEGMENTED BY HASH(a) ALL NODES;
=> CREATE TABLE T2 (x INT, y INT) SEGMENTED BY HASH(x) ALL NODES;
=> EXPLAIN SELECT * FROM T1 JOIN T2 ON T1.a = T2.y;

----- QUERY PLAN DESCRIPTION: -----
Access Path:
+-JOIN HASH [Cost: 639, Rows: 10K (NO STATISTICS)] (PATH ID: 1) Inner (RESEGMENT)
|   Join Cond: (T1.a = T2.y)
|   Materialize at Output: T1.b
|   Execute on: All Nodes
|   +-- Outer -> STORAGE ACCESS for T1 [Cost: 151, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
```

```
| |      Projection: public.T1_b0
| |      Materialize: T1.a
| |      Execute on: All Nodes
| +-- Inner -> STORAGE ACCESS for T2 [Cost: 302, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
| |      Projection: public.T2_b0
| |      Materialize: T2.x, T2.y
| |      Execute on: All Nodes
```

Analytic Function Path

Vertica attempts to optimize multiple SQL-99 [Analytic Functions](#) from the same query by grouping them together in Analytic Group areas.

For each analytical group, Vertica performs a distributed sort and resegment of the data, if necessary.

You can tell how many sorts and resegments are required based on the query plan.

For example, the following query plan shows that the [FIRST_VALUE](#) and [LAST_VALUE](#) functions are in the same analytic group because their `OVER` clause is the same. In contrast, `ROW_NUMBER()` has a different `ORDER BY` clause, so it is in a different analytic group. Because both groups share the same `PARTITION BY deal_stage` clause, the data does not need to be resegmented between groups :

```
EXPLAIN SELECT
  first_value(deal_size) OVER (PARTITION BY deal_stage
  ORDER BY deal_size),
  last_value(deal_size) OVER (PARTITION BY deal_stage
  ORDER BY deal_size),
  row_number() OVER (PARTITION BY deal_stage
  ORDER BY largest_bill_amount)
FROM customer_dimension;

Access Path:
+-ANALYTICAL [Cost: 1K, Rows: 50K] (PATH ID: 1)
| Analytic Group
|   Functions: row_number()
|   Group Sort: customer_dimension.deal_stage ASC, customer_dimension.largest_bill_amount ASC NULLS
LAST
| Analytic Group
|   Functions: first_value(), last_value()
|   Group Filter: customer_dimension.deal_stage
|   Group Sort: customer_dimension.deal_stage ASC, customer_dimension.deal_size ASC NULL LAST
|   Execute on: All Nodes
| +--> STORAGE ACCESS for customer_dimension [Cost: 263, Rows: 50K]
|   (PATH ID: 2)
| |      Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
| |      Materialize: customer_dimension.largest_bill_amount,
| |      customer_dimension.deal_stage, customer_dimension.deal_size
| |      Execute on: All Nodes
```

See Also

[Invoking Analytic Functions](#)

Node Down Information

Vertica provides performance optimization when cluster nodes fail by distributing the work of the down nodes uniformly among available nodes throughout the cluster.

When a node in your cluster is down, the query plan identifies which node the query will execute on. To help you quickly identify down nodes on large clusters, EXPLAIN output lists up to six nodes, if the number of running nodes is less than or equal to six, and lists only down nodes if the number of running nodes is more than six.

Note: The node that executes down node queries is not always the same one.

The following table provides more detail:

Node state	EXPLAIN output
If all nodes are up, EXPLAIN output indicates All Nodes.	Execute on: All Nodes
If fewer than 6 nodes are up, EXPLAIN lists up to six running nodes.	Execute on: [node_list].
If more than 6 nodes are up, EXPLAIN lists only non-running nodes.	Execute on: All Nodes Except [node_list]
If the node list contains non-ephemeral nodes, the EXPLAIN output indicates All Permanent Nodes.	Execute on: All Permanent Nodes
If the path is being run on the query initiator, the EXPLAIN output indicates Query Initiator.	Execute on: Query Initiator

Examples

In the following example, the down node is v_vmart_node0005, and the node v_vmart_node0006 will execute this run of the query.

```
=> EXPLAIN SELECT * FROM test;  
QUERY PLAN
```

```
-----  
QUERY PLAN DESCRIPTION:  
-----  
EXPLAIN SELECT * FROM my1table;  
Access Path:  
+-STORAGE ACCESS for my1table [Cost: 10, Rows: 2] (PATH ID: 1)  
| Projection: public.my1table_b0  
| Materialize: my1table.c1, my1table.c2  
| Execute on: All Except v_vmart_node0005  
+-STORAGE ACCESS for my1table (REPLACEMENT FOR DOWN NODE) [Cost: 66, Rows: 2]  
| Projection: public.my1table_b1  
| Materialize: my1table.c1, my1table.c2  
| Execute on: v_vmart_node0006
```

The `All Permanent Nodes` output in the following example fragment denotes that the node list is for permanent (non-ephemeral) nodes only:

```
=> EXPLAIN SELECT * FROM my2table;  
Access Path:  
+-STORAGE ACCESS for my2table [Cost: 18, Rows:6 (NO STATISTICS)] (PATH ID: 1)  
| Projection: public.my2table_b0  
| Materialize: my2table.x, my2table.y, my2table.z  
| Execute on: All Permanent Nodes
```

MERGE Path

Vertica prepares an optimized query plan for a [MERGE](#) statement if the statement and its tables meet the criteria described in [MERGE Optimization](#).

Use the [EXPLAIN](#) keyword to determine whether Vertica can produce an optimized query plan for a given [MERGE](#) statement. If optimization is possible, the [EXPLAIN](#)-generated output contains a `[Semi]` path, as shown in the following sample fragment:

```
...  
Access Path:  
+-DML DELETE [Cost: 0, Rows: 0]  
| Target Projection: public.A_b1 (DELETE ON CONTAINER)  
| Target Prep:  
| Execute on: All Nodes  
| +--> JOIN MERGEJOIN(inputs presorted) [Semi] [Cost: 6, Rows: 1 (NO STATISTICS)] (PATH ID: 1)  
| | Inner (RESEGMENT)  
| | | Join Cond: (A.a1 = VAL(2))  
| | | Execute on: All Nodes  
| | +-- Outer -> STORAGE ACCESS for A [Cost: 2, Rows: 2 (NO STATISTICS)] (PATH ID: 2)  
...
```

Conversely, if Vertica cannot create an optimized plan, [EXPLAIN](#)-generated output contains `RightOuter` path:

```
...  
Access Path: +-DML MERGE
```

```
| Target Projection: public.locations_b1
| Target Projection: public.locations_b0
| Target Prep:
| Execute on: All Nodes
| +---> JOIN MERGEJOIN(inputs presorted) [RightOuter] [Cost: 28, Rows: 3 (NO STATISTICS)] (PATH ID:
1) Outer (RESEGMENT) Inner (RESEGMENT)
| |      Join Cond: (locations.user_id = VAL(2)) AND (locations.location_x = VAL(2)) AND
(locations.location_y = VAL(2))
| |      Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for <No Alias> [Cost: 15, Rows: 2 (NO STATISTICS)] (PATH ID: 2)
...

```

Directed Queries

Directed queries encapsulate information that the optimizer can use to create a query plan. Directed queries can serve the following goals:

- Preserve current query plans before a scheduled upgrade. In most instances, queries perform more efficiently after a Vertica upgrade. In the few cases where this is not so, you can use directed queries that you created before upgrading, to recreate query plans from the earlier version.
- Enable you to create query plans that improve optimizer performance. Occasionally, you might want to influence the optimizer to make better choices in executing a given query. For example, you can choose a different projection, or force a different join order. In this case, you can use a directed query to create a query plan that preempts any plan that the optimizer might otherwise create.
- Redirect an input query to a query that uses different semantics—for example, map a join query to a SELECT statement that queries a flattened table.

Directed Query Components

A directed query pairs two components:

- **Input query:** A query that triggers use of this directed query when it is active.
- **Annotated query:** A SQL statement with embedded optimizer hints, which instruct the optimizer how to create a query plan for the specified input query. These hints specify important query plan elements, such as join order and projection choices.

You can also use most optimizer hints directly in vsql. For information about these and other hints, see [Hints](#) in the SQL Reference Manual.

Vertica provides two methods for creating directed queries:

- The optimizer can generate an annotated query from a given input query and pair the two as a directed query.
- You can write your own annotated query and pair it with an input query.

For a description of both methods, see [Creating Directed Queries](#).

Creating Directed Queries

`CREATE DIRECTED QUERY` associates an input query with a query annotated with optimizer hints. It stores the association under a unique identifier. `CREATE DIRECTED QUERY` has two variants:

- `CREATE DIRECTED QUERY OPTIMIZER` directs the query optimizer to generate annotated SQL from the specified input query. The annotated query contains hints that the optimizer can use to recreate its current query plan for that input query.
- `CREATE DIRECTED QUERY CUSTOM` specifies an annotated query supplied by the user. Vertica associates the annotated query with the input query specified by the last `SAVE QUERY` statement.

In both cases, Vertica associates the annotated query and input query, and registers their association in the system table `V_CATALOG.DIRECTED_QUERIES` under `query_name`.

[The two approaches can be used together](#): you can use the annotated SQL that the optimizer creates as the basis for creating your own (custom) directed queries.

Optimizer-Generated Directed Queries

`CREATE DIRECTED QUERY OPTIMIZER` passes an input query to the optimizer, which generates an annotated query from its own query plan. It then pairs the input and annotated queries and saves them as a directed query.

Note: The input query that you supply for optimizer-generated directed queries supports only one optimizer hint, `IGNORECONST`.

You can use optimized-generated directed queries to capture query plans before you upgrade. Doing so can be especially useful if you detect diminished performance of a given query after

the upgrade. In this case, you can use the corresponding directed query to recreate an earlier query plan, and compare its performance to the plan generated by the current optimizer.

For example, the following SQL statements create and activate the directed query `findBostonCashiers_OPT`:

```
=> CREATE DIRECTED QUERY OPTIMIZER 'findBostonCashiers_OPT'
    SELECT employee_first_name, employee_last_name FROM public.employee_dimension
        WHERE employee_city='Boston' and job_title='Cashier';
CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
ACTIVATE DIRECTED QUERY
```

After this directed query plan is activated, the optimizer uses it to generate a query plan for all subsequent invocations of its input query. You can view the optimizer-generated annotated query by either calling [GET DIRECTED QUERY](#) or querying the system table [V_CATALOG.DIRECTED_QUERIES](#):

```
=> SELECT query_name, annotated_query FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name =
'findBostonCashiers_OPT';
-[ RECORD 1 ]-----+-----
query_name      | findBostonCashiers_OPT
annotated_query | SELECT /*+ verbatim */ employee_dimension.employee_first_name AS employee_first_
name, employee_dimension.employee_last_name AS employee_last_name
FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/
WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title =
'Cashier'::varchar(7))
```

The annotated SQL includes two hints:

- `/*+verbatim*/` specifies to execute the annotated query exactly as written and produce a query plan accordingly.
- `/*+projs('public.Emp_Dimension')*/` directs the optimizer to create a query plan that uses the projection `public.Emp_Dimension`.

The following EXPLAIN statement verifies the optimizer's use of this directed query and the specified projection:

```
QUERY PLAN DESCRIPTION:
-----
EXPLAIN SELECT employee_first_name, employee_last_name FROM employee_dimension WHERE employee_
city='Boston' AND job_title='Cashier';

The following active directed query(query name: findBostonCashiers_OPT) is being executed:
SELECT /*+verbatim*/ employee_dimension.employee_first_name, employee_dimension.employee_last_name
FROM public.employee_dimension employee_dimension/*+projs('public.employee_dimension')*/ WHERE
((employee_dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title =
'Cashier'::varchar(7)))

Access Path:
```

```
+--STORAGE ACCESS for employee_dimension [Cost: 60, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.employee_dimension_b0
| Materialize: employee_dimension.employee_first_name, employee_dimension.employee_last_name
| Filter: (employee_dimension.employee_city = 'Boston')
| Filter: (employee_dimension.job_title = 'Cashier')
| Execute on: All Nodes
```

Custom Directed Queries

`CREATE DIRECTED QUERY CUSTOM` specifies an annotated query and pairs it to an input query previously saved by `SAVE QUERY`. The `SAVE QUERY` statement must precede `CREATE DIRECTED QUERY CUSTOM`. `SAVE QUERY` temporarily saves an input query for use in creating a directed query. You must issue both statements in the same user session.

Note: The input query that you supply to `SAVE QUERY` supports only one optimizer hint, `IGNORECONST`.

In the following example, `SAVE QUERY` saves a query. The `CREATE DIRECTED QUERY CUSTOM` statement that follows it provides an annotated query that includes a `/*+projs*/` hint. This hint instructs the optimizer to use the projection `public.Emp_Dimension_Unseg`:

```
=> SAVE QUERY SELECT employee_first_name, employee_last_name FROM employee_dimension
    WHERE employee_city='Boston' AND job_title='Cashier';
SAVE QUERY

=> CREATE DIRECTED QUERY CUSTOM 'findBostonCashiers_CUSTOM'
    SELECT employee_first_name, employee_last_name
    FROM employee_dimension /*+Projs('public.emp_dimension_unseg')*/
    WHERE employee_city='Boston' AND job_title='Cashier';
CREATE DIRECTED QUERY
```

Caution: Vertica associates a saved query and annotated query without checking whether the input and annotated queries are compatible. Be careful to sequence `SAVE QUERY` and `CREATE DIRECTED QUERY CUSTOM` so the saved and directed queries are correctly matched.

After this directed query plan is activated, the optimizer uses it to generate a query plan for all subsequent invocations of its input query. The following `EXPLAIN` output verifies the optimizer's use of this directed query and the projection it specifies:

```
=> DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
DEACTIVATE DIRECTED QUERY
=> ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
ACTIVATE DIRECTED QUERY
```

```
=> EXPLAIN SELECT employee_first_name, employee_last_name FROM employee_dimension
    WHERE employee_city='Boston' AND job_title='Cashier';

QUERY PLAN
-----
QUERY PLAN DESCRIPTION:
-----
EXPLAIN SELECT employee_first_name, employee_last_name FROM employee_dimension where employee_
city='Boston' AND job_title='Cashier';

The following active directed query(query name: findBostonCashiers_CUSTOM) is being executed:
SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name FROM
public.employee_dimension/*+Projs('public.Emp_Dimension_Unseg')*/ WHERE ((employee_
dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title =
'Cashier'::varchar(7)))

Access Path:
+-STORAGE ACCESS for employee_dimension [Cost: 158, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.emp_dimension_unseg
| Materialize: employee_dimension.employee_first_name, employee_dimension.employee_last_name
| Filter: (employee_dimension.employee_city = 'Boston')
| Filter: (employee_dimension.job_title = 'Cashier')
| Execute on: Query Initiator
```

Using Optimizer-Generated and Custom Directed Queries Together

You can use the annotated SQL that the optimizer creates as the basis for creating your own (custom) directed queries. This approach can be especially useful in evaluating the plan that the optimizer creates to handle a given query, and testing plan modifications.

For example, you might want to modify how the optimizer implements the following query:

```
=> SELECT customers.name FROM
(purchase JOIN customers ON customers.customer_id = purchase.customer_id)
JOIN item ON purchase.item_id = item.item_id
WHERE item.type = 'household' AND customers.age < 30 AND MONTH(purchase.date) = 'March';
```

When you run EXPLAIN on this query, you discover that the optimizer uses projection `customers_proj_age` for the `customers` table. This projection is sorted on the column `customers.age`. Consequently, the optimizer hash-joins the tables `customers` and `purchase` on `customer_id`.

After analyzing the `customers` table data, you make two observations:

- Most customers are under 30, so it makes more sense to use a different projection, `customers_proj_id`, which is sorted on customer IDs.
- Because the inner join projections are both sorted on customer IDs, the query plan should also specify a merge join.

You can create a directed query that encapsulates these changes as follows:

1. Create an optimizer-generated directed query from the input query, by using **CREATE DIRECTED QUERY OPTIMIZER**:

```
=> CREATE DIRECTED QUERY OPTIMIZER totalStoreSales SELECT customers.name FROM
  (purchase JOIN customers ON customers.customer_id = purchase.customer_id)
  JOIN item ON purchase.item_id = item.item_id)
  WHERE item.type = 'household' AND customers.age < 30 AND month(purchase.date) = 'March';
CREATE DIRECTED QUERY
```

Note: **CREATE DIRECTED QUERY OPTIMIZER** provides a convenient short cut for creating syntactically correct annotated queries that you can edit. For full information about supported optimizer hints, see [Hints](#) in the SQL Reference Manual.

2. Get the annotated query generated for this directed query: call **GET DIRECTED QUERY** or query the system table **V_CATALOG.DIRECTED_QUERIES**. The following example queries the system table:

```
=> SELECT annotated_query FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name='totalStoreSales';
-[ RECORD 1 ]---+-----
annotated_query | SELECT /*+ SYNTACTIC_JOIN,VERBATIM */ customers.name
FROM ((purchase /*+PROJS('purchase_cid')*/
JOIN /*+ JTYPE(H) */ customers /*+PROJS('customers_proj_age')*/
ON customers.customer_id = purchase.customer_id)
JOIN /*+ JTYPE(H) */ item /*+PROJS('item_proj_type')*/
ON purchase.item_id = item.item_id)
WHERE item.type = 'household' AND customers.age < 30 AND month(purchase.date) = 'March'
```

3. Modify the annotated query:

```
SELECT /*+ SYNTACTIC_JOIN,VERBATIM */ customers.name
FROM ((purchase /*+PROJS('purchase_cid')*/
JOIN /*+ JTYPE(M) */ customers /*+PROJS('customers_proj_id')*/
ON customers.customer_id = purchase.customer_id)
JOIN /*+ JTYPE(H) */ item /*+PROJS('item_proj_type')*/
ON purchase.item_id = item.item_id)
WHERE item.type = 'household' AND customers.age < 30 AND month(purchase.date) = 'March'
```

4. Use the modified annotated query to create the desired directed query:
 - Save the desired input query with **SAVE QUERY**:

```
=> SAVE QUERY customers.name FROM
  (purchase JOIN customers ON customers.customer_id = purchase.customer_id)
  JOIN item ON purchase.item_id = item.item_id)
  WHERE item.type = 'household' AND customers.age < 30 AND month(purchase.date) = 'March';
SAVE QUERY
```

- Create a custom directed query that associates the saved input query with the modified annotated query:

```
=> CREATE DIRECTED QUERY CUSTOM 'getCustomersUnder30'  
SELECT /*+ SYNTACTIC_JOIN,VERBATIM */ customers.name FROM  
((purchase /*+PROJS('purchase_cid')*/  
  JOIN /*+ JTYPE(M) */ customers /*+PROJS('customers_proj_id')*/  
  ON customers.customer_id = purchase.customer_id)  
 JOIN /*+ JTYPE(H) */ item /*+PROJS('item_proj_type')*/ ON purchase.item_id = item.item_id)  
WHERE item.type = 'household' AND customers.age < 30 AND month(purchase.date) = 'March'  
CREATE DIRECTED QUERY
```

5. Activate this directed query:

```
ACTIVATE DIRECTED QUERY getCustomersUnder30;  
ACTIVATE DIRECTED QUERY
```

When the optimizer processes a query that matches this directed query's input query, it uses the directed query's annotated query to generate a query plan.

Setting Hints in Annotated Queries

The hints in a directed query's annotated query provide the Vertica optimizer instructions how to execute an input query. Annotated queries support the following hints:

- **Join hints** specify join order, join type, and join data distribution: [SYNTACTIC_JOIN](#), [DISTRIB](#), [JTYPE](#), [UTYPE](#).
- **Table hints** specify which projections to include and exclude in the query plan: [PROJS](#), [SKIP_PROJS](#).
- [IGNORECONST](#) is supported for a directed query's input and annotated queries. For more information, see [Ignoring Constants in Directed Queries](#)
- [VERBATIM](#) enforces execution of an annotated query exactly as written.

Other hints in annotated queries such as [DIRECT](#) or [LABEL](#) are ignored.

You can use hints in a vsql query the same as in an annotated query, with two exceptions: [IGNORECONST](#) and [VERBATIM](#) are invalid for use in vsql. For general information about using hints, see [Hints](#) in the SQL Reference Manual.

Ignoring Constants in Directed Queries

The `IGNORECONST` hint lets you create directed queries that support input queries with various conditions. For example, you might want to use the same directed query for the following input queries:

```
=> SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension WHERE  
Employee_city='Boston' and Employee_position='Cashier';  
  
=> SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension WHERE  
Employee_city='Chicago' and Employee_position='Greeter';
```

In this case, you can create a directed query where input and annotated queries qualify the settings for `Employee_city` and `Employee_position` with `IGNORECONST` hints:

```
=> SAVE QUERY SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension  
WHERE Employee_city='somewhere' /*+IGNORECONST(1)*/  
AND Employee_position='somejob' /*+IGNORECONST(2)*/;  
SAVE QUERY  
  
=> CREATE DIRECTED QUERY CUSTOM 'findEmployees'  
SELECT Employee_first_name, Employee_last_name  
FROM EMP_Dimension /*+projs('public.Emp_Dimension_Unseg')*/  
WHERE Employee_city='somewhere' /*+IGNORECONST(1)*/  
AND Employee_position='somejob' /*+IGNORECONST(2)*/  
CREATE DIRECTED QUERY  
  
=> ACTIVATE DIRECTED QUERY findEmployees;  
ACTIVATE DIRECTED QUERY
```

`IGNORECONST` requires an integer argument. This argument matches constants in input and annotated queries that you want the optimizer to ignore. In the previous example, the input and annotated queries of the directed query, `findEmployees`, use `IGNORECONST` to pair two sets of constants:

- `IGNORECONST(1)` pairs input and annotated query settings for `Employee_city`.
- `IGNORECONST(2)` pairs input and annotated query settings for `Employee_position`.

When the optimizer maps input queries to the directed query `findEmployees`, the `IGNORECONST` arguments for `Employee_city` and `Employee_position` tell it to ignore the saved values for these two columns. Thus, users can supply any values for these columns.

For example, the following query plan shows how the optimizer maps one user query to the directed query `findEmployees`:

```
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension WHERE Employee_
city='Boston' AND Employee_position='Cashier';

The following active directed query is being executed:
SELECT EMP_Dimension.Employee_first_name, EMP_Dimension.Employee_last_name FROM public.Emp_Dimension
EMP_Dimension/*+projs('public.Emp_Dimension_Unseg')*/ WHERE (EMP_Dimension.Employee_city =
'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position = 'Cashier'::varchar(7))

Access Path:
+-STORAGE ACCESS for EMP_Dimension [Cost: 154, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.Emp_Dimension_Unseg
| Materialize: EMP_Dimension.Employee_first_name, EMP_Dimension.Employee_last_name
| Filter: (EMP_Dimension.Employee_city = 'Boston')
| Filter: (Emp_Dimension.Employee_position = 'Cashier')
| Execute on: Query Initiator
```

Embedding IGNORECONST Hints in Optimizer-Generated Directed Queries

You can embed `IGNORECONST` hints in the input query argument of `CREATE DIRECTED QUERY OPTIMIZER`. The optimizer creates an annotated query that includes `IGNORECONST` hints for the corresponding columns. For example, given this `CREATE DIRECTED QUERY OPTIMIZER` statement:

```
=> CREATE DIRECTED QUERY OPTIMIZER findGreetersAnyCity
SELECT Employee_first_name, Employee_last_name
FROM EMP_Dimension /*+projs('public.Emp_Dimension_Unseg')*/
WHERE Employee_city='anywhere'/*+IGNORECONST(9)*/ AND Employee_position='Greeter';
CREATE DIRECTED QUERY
```

the optimizer creates the following annotated query:

```
SELECT /*+ verbatim */ EMP_Dimension.Employee_first_name AS Employee_first_name, EMP_
Dimension.Employee_last_name AS Employee_last_name
FROM public.Emp_Dimension AS EMP_Dimension/*+projs('public.Emp_Dimension_Unseg')*/
WHERE (EMP_Dimension.Employee_city = 'anywhere'::varchar(8) /*+IgnoreConst(9)*/) AND (EMP_
Dimension.Employee_position = 'Greeter'::varchar(7))
```

Mapping One-to-Many IGNORECONST Hints

The examples shown so far demonstrate one-to-one pairings of IGNORECONST hints. You can also use IGNORECONST to map one input constant to multiple constants in an annotated query. This approach can be especially useful when you want to provide the optimizer with explicit instructions how to execute a query that joins tables.

For example, this simple query joins two tables:

```
SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 8;
```

In this case, the optimizer can infer that `S.a` and `T.b` have the same value and implements the join accordingly. In other cases, you might want to provide the optimizer explicit guidance on implementing a join condition. Given the previous input query, you can create a directed query that uses the IGNORECONST hint to map query input for `S.a` to `S.a` and `T.b`:

```
=> SAVE QUERY SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 999/*+IGNORECONST(1)*/;  
SAVE QUERY  
  
=> CREATE DIRECTED QUERY CUSTOM joinST SELECT * FROM S JOIN T ON S.a = T.b  
WHERE S.a = 999/*+IGNORECONST(1)*/ AND T.b = 999/*+IGNORECONST(1)*/;  
CREATE DIRECTED QUERY  
  
=> ACTIVATE DIRECTED QUERY joinST;  
ACTIVATE DIRECTED QUERY
```

Now, given the following input query:

```
SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 3;
```

the optimizer uses the directed query `joinST` and its IGNORECONST hints to rewrite the query as follows:

```
QUERY PLAN DESCRIPTION:  
-----  
  
EXPLAIN SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 3;  
  
The following active directed query(query name: joinST) is being executed:  
SELECT S.a, T.b FROM (public.S JOIN public.T ON ((S.a = T.b))) WHERE ((S.a = 3) AND (T.b = 3))  
  
Access Path:  
+-JOIN MERGEJOIN(inputs presorted) [Cost: 11, Rows: 9 (NO STATISTICS)] (PATH ID: 1)  
|   Join Cond: (S.a = T.b)  
|   Execute on: v_vmart_node0002  
| +- Outer -> STORAGE ACCESS for S [Cost: 5, Rows: 9 (NO STATISTICS)] (PATH ID: 2)  
| |   Projection: public.S_b0  
| |   Materialize: S.a  
| |   Filter: (S.a = 3)
```

```
| |      Execute on: v_vmart_node0002
| |      Runtime Filter: (SIP1(MergeJoin): S.a)
| +-- Inner -> STORAGE ACCESS for T [Cost: 5, Rows: 9 (NO STATISTICS)] (PATH ID: 3)
| |      Projection: public.T_b0
| |      Materialize: T.b
| |      Filter: (T.b = 3)
| |      Execute on: v_vmart_node0002
```

Rewriting Queries

You can use directed queries to change the semantics of a given query—that is, substitute one query for another. This can be especially important when you have little or no control over the content and format of input queries that your Vertica database processes. You can map these queries to directed queries that rewrite the original input for optimal execution.

The following sections describe two use cases:

- [Rewriting Join Queries](#)
- [Using Query Templates](#)

Rewriting Join Queries

Many of your input queries join multiple tables. With the recent introduction of [flattened tables](#), you've determined that in many cases, it would be more efficient to denormalize much of your data in several wide tables and query those tables directly. You cannot revise the input queries themselves. However, you can use directed queries to map these queries to queries on the flattened table data.

For example, the following query aggregates regional sales of white wine products, by joining three tables in the VMart database:

```
=> SELECT SD.store_region AS 'Sales Region',
       SD.store_city AS 'City',
       SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
```

You can consolidate the joined table data in a single flattened table, and query this table instead. By doing so, you can access the same data faster. You can create this table with the following SQL:

```
CREATE TABLE store.store_sales_wide AS SELECT * from store.store_sales_fact;
ALTER TABLE store.store_sales_wide ADD COLUMN store_name VARCHAR(64)
  SET USING (SELECT store_name FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide ADD COLUMN store_city varchar(64)
  SET USING (SELECT store_city FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide ADD COLUMN store_state char(2)
  SET USING (SELECT store_state char FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide ADD COLUMN store_region varchar(64)
  SET USING (SELECT store_region FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide add column product_description VARCHAR(128)
  SET USING (SELECT product_description FROM public.product_dimension
  WHERE store_sales_wide.product_key||store_sales_wide.product_version = product_dimension.product_
  key||product_dimension.product_version);
ALTER TABLE store.store_sales_wide ADD COLUMN sku_number char(32)
  SET USING (SELECT sku_number char FROM product_dimension
  WHERE store_sales_wide.product_key||store_sales_wide.product_version = product_dimension.product_
  key||product_dimension.product_version);

SELECT REFRESH_COLUMNS ('store.store_sales_wide','', 'rebuild');
```

After creating this table and refreshing its SET USING columns, you can rewrite the earlier query as follows:

```
=> SELECT store_region AS 'Sales Region',
  store_city AS 'City',
  SUM(gross_profit_dollar_amount) AS Total
  FROM store.store_sales_wide
  WHERE product_description ILIKE '%wine%' AND product_description ILIKE '%white%'
  GROUP BY ROLLUP (store_region, store_city)
  ORDER BY 1,3 DESC;
```

Sales Region	City	Total
East		762632
East	Sterling Heights	67391
East	Allentown	64528
East	Elizabeth	58784
East	New Haven	57730
East	Boston	57315
East	Lowell	42734
East	Charlotte	40757
East	Erie	39070
East	Washington	38792
East	Waterbury	35369
East	Hartford	27684
East	Clarksville	25953
East	Stamford	25657
East	Memphis	25629
East	Baltimore	23999

East	Columbia	22573
East	Manchester	22496
East	Nashville	22353
East	Cambridge	14385
East	Philadelphia	13017
East	Alexandria	12273
East	New York	12261
East	Portsmouth	11882
MidWest		494182
MidWest	Lansing	69157
MidWest	Livonia	62269
...		

Querying the flattened table is more efficient; however, you still must account for input queries that continue to use the earlier join syntax. You can do so by creating a [custom directed query](#), which redirects these input queries to the desired syntax:

1. [Save the input query](#):

```
=> SAVE QUERY SELECT SD.store_region AS 'Sales Region',
    SD.store_city AS 'City',
    SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_
version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
SAVE QUERY
```

2. [Map the saved query to a directed query with the desired syntax, and activate the directed query](#):

```
=> CREATE DIRECTED QUERY CUSTOM 'RegionalSalesWhiteWine'
    SELECT store_region AS 'Sales Region',
    store_city AS 'City',
    SUM(gross_profit_dollar_amount) AS Total
FROM store.store_sales_wide
WHERE product_description ILIKE '%wine%' AND product_description ILIKE '%white%'
GROUP BY ROLLUP (store_region, store_city)
ORDER BY 1,3 DESC;
CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY RegionalSalesWhiteWine;
ACTIVATE DIRECTED QUERY
```

When directed query `RegionalSalesWhiteWine` is active, the query optimizer maps all queries that match the original input format to the directed query, as shown in the following query plan:

```
=> EXPLAIN SELECT SD.store_region AS 'Sales Region',
      SD.store_city AS 'City',
      SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
```

The following active directed query(query name: RegionalSalesWhiteWine) is being executed:

```
SELECT store_sales_wide.store_region AS "Sales Region",
      store_sales_wide.store_city AS City,
      sum(store_sales_wide.gross_profit_dollar_amount) AS Total FROM store.store_sales_wide
WHERE ((store_sales_wide.product_description ~* '%wine%'::varchar(6))
AND (store_sales_wide.product_description ~* '%white%'::varchar(7)))
GROUP BY GROUPING SETS((store_sales_wide.store_region, store_sales_wide.store_city), (store_sales_
wide.store_region), ())
ORDER BY store_sales_wide.store_region, sum(store_sales_wide.gross_profit_dollar_amount) DESC
```

Access Path:

```
+--SORT [Cost: 76K, Rows: 100] (PATH ID: 1)
| Order: store_sales_wide.store_region ASC, sum(store_sales_wide.gross_profit_dollar_amount) DESC
| Execute on: All Nodes
| +---> GROUPBY HASH (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 76K, Rows: 100] (PATH
ID: 2)
| | Aggregates: sum(store_sales_wide.gross_profit_dollar_amount)
| | Group By: store_sales_wide.store_region, store_sales_wide.store_city
| | Grouping Sets: (store_sales_wide.store_region, store_sales_wide.store_city, <SVAR>),
(store_sales_wide.store_region, <SVAR>), (<SVAR>)
| | Execute on: All Nodes
| | +---> STORAGE ACCESS for store_sales_wide [Cost: 23K, Rows: 2M] (PATH ID: 3)
| | | Projection: store.store_sales_wide_super
| | | Materialize: store_sales_wide.gross_profit_dollar_amount, store_sales_wide.store_city,
store_sales_wide.store_region
| | | Filter: ((store_sales_wide.product_description ~* '%wine%') AND (store_sales_
wide.product_description ~* '%white%'))
| | | Execute on: All Nodes
```

To compare the costs of executing the directed query and executing the original input query, deactivate the directed query and use EXPLAIN on the original input query. The optimizer reverts to creating a plan for the input query that incurs significantly greater cost, as shown in the following query plan:

```
=> DEACTIVATE DIRECTED QUERY RegionalSalesWhiteWine;
DEACTIVATE DIRECTED QUERY

=> EXPLAIN SELECT SD.store_region AS 'Sales Region',
      SD.store_city AS 'City',
      SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
```

```

Access Path:
+-SORT [Cost: 192K, Rows: 100] (PATH ID: 1)
| Order: SD.store_region ASC, sum(SF.gross_profit_dollar_amount) DESC
| Execute on: All Nodes
| +---> GROUPBY HASH (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 192K, Rows: 100]
(PATH ID: 2)
| | Aggregates: sum(SF.gross_profit_dollar_amount)
| | Group By: SD.store_region, SD.store_city
| | Grouping Sets: (SD.store_region, SD.store_city, <SVAR>), (SD.store_region, <SVAR>),
(<SVAR>)
| | Execute on: All Nodes
| | +---> JOIN HASH [Cost: 17K, Rows: 5M] (PATH ID: 3) Inner (BROADCAST)
| | | Join Cond: (concat((SF.product_key)::varchar, (SF.product_version)::varchar) = concat
((P.product_key)::varchar, (P.product_version)::varchar))
| | | Materialize at Input: SF.product_key, SF.product_version
| | | Materialize at Output: SF.gross_profit_dollar_amount
| | | Execute on: All Nodes
| | | +-- Outer -> JOIN HASH [Cost: 5K, Rows: 5M] (PATH ID: 4) Inner (BROADCAST)
| | | | Join Cond: (SF.store_key = SD.store_key)
| | | | Execute on: All Nodes
| | | +-- Outer -> STORAGE ACCESS for SF [Cost: 4K, Rows: 5M] (PATH ID: 5)
| | | | Projection: store.store_sales_fact_super
| | | | Materialize: SF.store_key
| | | | Execute on: All Nodes
| | | | Runtime Filters: (SIP2(HashJoin): SF.store_key), (SIP1(HashJoin): concat((SF.product_
key)::varchar, (SF.product_version)::varchar))
| | | +-- Inner -> STORAGE ACCESS for SD [Cost: 49, Rows: 250] (PATH ID: 6)
| | | | Projection: store.store_dimension_super
| | | | Materialize: SD.store_key, SD.store_city, SD.store_region
| | | | Execute on: All Nodes
| | | +-- Inner -> STORAGE ACCESS for P [Cost: 378, Rows: 60K] (PATH ID: 7)
| | | | Projection: public.product_dimension_super
| | | | Materialize: P.product_key, P.product_version
| | | | Filter: ((P.product_description ~* '%wine%') AND (P.product_description ~*
'%white%'))
| | | | Execute on: All Nodes

```

Using Query Templates

You can use directed queries to implement multiple queries that are identical except for the predicate strings on which query results are filtered. For example, directed query `RegionalSalesWhiteWine` only handles input queries that filter on `product_description` values containing the strings `wine` and `white`. You can create a modified version of this directed query that matches the syntax of multiple input queries, which differ only in their pairs of input values—for example, `wine` and `red`.

You create this query template in the following steps:

1. Use the input query on the flattened table to create an [optimized-generated directed query](#):

```
=> CREATE DIRECTED QUERY optimizer RegionalSalesTemp
  SELECT store_region AS 'Sales Region',
         store_city AS 'City',
         SUM(gross_profit_dollar_amount) AS Total
  FROM store.store_sales_wide
  WHERE product_description ILIKE '%wine%'
         AND product_description ILIKE '%white%'
  GROUP BY ROLLUP (store_region, store_city)
  ORDER BY 1,3 DESC;
CREATE DIRECTED QUERY
```

2. Modify the original join input query with [IGNORECONST](#) hints and [save it](#):

```
=> SAVE QUERY SELECT SD.store_region AS 'Sales Region',
  SD.store_city AS 'City',
  SUM(SF.gross_profit_dollar_amount) Total
  FROM store.store_sales_fact SF
  JOIN store.store_dimension SD ON SF.store_key=SD.store_key
  JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_
  version
  WHERE P.product_description ILIKE 'desc-string1' /*+IGNORECONST(1)*/
         AND P.product_description ILIKE 'desc-string2' /*+IGNORECONST(2)*/
  GROUP BY ROLLUP (SD.store_region, SD.store_city)
  ORDER BY 1,3 DESC;
SAVE QUERY
```

3. [Get the optimizer-generated query](#) from system table [DIRECTED_QUERIES](#):

```
=> \x
=> SELECT annotated_query FROM DIRECTED_QUERIES WHERE query_name='RegionalSalesTemp';
-[ RECORD 1 ]-----+-----
annotated_query | SELECT /*+verbatim*/ store_sales_wide.store_region AS "Sales Region",
  store_sales_wide.store_city AS City,
  sum(store_sales_wide.gross_profit_dollar_amount) AS Total
  FROM store.store_sales_wide AS store_sales_wide/*+projs('store.store_sales_wide')*/
  WHERE ((store_sales_wide.product_description ~* '%wine% '::varchar(6))
         AND (store_sales_wide.product_description ~* '%white% '::varchar(7)))
  GROUP BY /*+GByType(Hash)*/ GROUPING SETS((1, 2), (1), ())
  ORDER BY 1 ASC, 3 DESC
(1 row)
```

4. Create a modified version of the annotated query that qualifies the input values with [IGNORECONST](#) hints. Use this version to create a custom directed query, and then activate it:

```
=> CREATE DIRECTED QUERY CUSTOM RegionalSales
  SELECT /*+verbatim*/ store_sales_wide.store_region AS "Sales Region",
         store_sales_wide.store_city AS City,
         sum(store_sales_wide.gross_profit_dollar_amount) AS Total
  FROM store.store_sales_wide AS store_sales_wide/*+projs('store.store_sales_wide')*/
  WHERE ((store_sales_wide.product_description ~* 'desc-str'/*+IGNORECONST(1)*/)
         AND (store_sales_wide.product_description ~* 'desc-str'/*+IGNORECONST(2)*/))
```

```
GROUP BY /*+GByType(Hash)*/ GROUPING SETS((1, 2), (1), ())
ORDER BY 1 ASC, 3 DESC;
CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY RegionalSales;
```

After activating this directed query, Vertica can use it for input queries that match the template. These queries can use any pair of strings to filter the result set. For example, the following input query filters on the strings `chicken` and `frozen`:

```
EXPLAIN SELECT SD.store_region AS 'Sales Region',
  SD.store_city AS 'City',
  SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%chicken%' AND P.product_description ILIKE '%frozen%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
```

The following active directed query(query name: `RegionalSales`) is being executed:

```
SELECT /*+verbatim*/ store_sales_wide.store_region AS "Sales Region",
  store_sales_wide.store_city AS City,
  sum(store_sales_wide.gross_profit_dollar_amount) AS Total
FROM store.store_sales_wide store_sales_wide/*+projs('store.store_sales_wide')*/
WHERE ((store_sales_wide.product_description ~* '%chicken%'::varchar(9))
  AND (store_sales_wide.product_description ~* '%frozen%'::varchar(8)))
GROUP BY /*+GByType(Hash)*/ GROUPING SETS((store_sales_wide.store_region, store_sales_wide.store_
city), (store_sales_wide.store_region), ())
ORDER BY store_sales_wide.store_region, sum(store_sales_wide.gross_profit_dollar_amount) DESC
```

Access Path:

```
+--SORT [Cost: 214K, Rows: 100] (PATH ID: 1)
| Order: store_sales_wide.store_region ASC, sum(store_sales_wide.gross_profit_dollar_amount) DESC
| Execute on: All Nodes
| +---> GROUPBY HASH (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 214K, Rows: 100]
(PATH ID: 2)
| | Aggregates: sum(store_sales_wide.gross_profit_dollar_amount)
| | Group By: store_sales_wide.store_region, store_sales_wide.store_city
| | Grouping Sets: (store_sales_wide.store_region, store_sales_wide.store_city, <SVAR>),
(store_sales_wide.store_region, <SVAR>), (<SVAR>)
| | Execute on: All Nodes
| | +---> STORAGE ACCESS for store_sales_wide [Cost: 39K, Rows: 5M] (PATH ID: 3)
| | | Projection: store.store_sales_wide_super
| | | Materialize: store_sales_wide.gross_profit_dollar_amount, store_sales_wide.store_city,
store_sales_wide.store_region
| | | Filter: ((store_sales_wide.product_description ~* '%chicken%') AND (store_sales_
wide.product_description ~* '%frozen%'))
| | | Execute on: All Nodes
```

When you execute this query, it returns with the following results:

Sales Region	City	Total
East		1421528

East	Sterling Heights	122205
East	Elizabeth	121061
East	Boston	113776
East	Allentown	109808
East	Lowell	101147
East	New Haven	95566
East	Waterbury	76076
East	Erie	68937
East	Charlotte	68032
East	Washington	64503
East	Clarksville	50772
East	Columbia	48207
East	Memphis	47014
East	Stamford	46150
East	Baltimore	45588
East	Nashville	44868
East	Manchester	43682
East	Hartford	43605
East	Philadelphia	24460
East	New York	21851
East	Alexandria	21794
East	Cambridge	21750
East	Portsmouth	20676
MidWest		937225
MidWest	Lansing	130754
MidWest	Livonia	121409
...		

Managing Directed Queries

Vertica provides a number of ways to manage directed queries:

- [Getting Directed Queries](#)
- [Identifying Active Directed Queries](#)
- [Activating and Deactivating Directed Queries](#)
- [Exporting Directed Queries from the Catalog](#)
- [Dropping Directed Queries](#)

Getting Directed Queries

You can obtain catalog information about directed queries in two ways:

- [Run the statement](#) `GET DIRECTED QUERY.`
- [Query the system table](#) `V_CATALOG.DIRECTED_QUERIES.`

Run GET DIRECTED QUERY

`GET DIRECTED QUERY` queries the system table `V_CATALOG.DIRECTED_QUERIES` on the specified input query. It returns a list of directed queries that map to the input query.

The following `GET DIRECTED QUERY` statement returns two directed queries that map to the same input query, `findBostonCashiers_OPT` and `findBostonCashiers_CUSTOM`:

```
=> GET DIRECTED QUERY SELECT employee_first_name, employee_last_name
    FROM employee_dimension WHERE employee_city='Boston' AND job_title='Cashier';
-[ RECORD 1 ]---+
query_name      | findBostonCashiers_OPT
is_active       | f
vertica_version | Vertica Analytic Database v7.2.3
comment         | Optimizer-generated directed query
creation_date   | 2016-04-25 08:17:26.913339
annotated_query | SELECT /*+ verbatim */ employee_dimension.employee_first_name AS employee_first_
name, employee_dimension.employee_last_name AS employee_last_name
FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/
WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title =
'Cashier'::varchar(7))
-[ RECORD 2 ]---+
query_name      | findBostonCashiers_CUSTOM
is_active       | t
vertica_version | Vertica Analytic Database v7.2.3
comment         | Custom directed query
creation_date   | 2016-04-25 09:15:11.464417
annotated_query | SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_
name FROM public.employee_dimension/*+Projs('public.Emp_Dimension_Unseg')*/ WHERE ((employee_
dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title =
'Cashier'::varchar(7)))
```

Query V_CATALOG.DIRECTED_QUERIES

You can query the system table `V_CATALOG.DIRECTED_QUERIES` directly. For example:

```
=> SELECT query_name, is_active FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name ILIKE
'%findBostonCashiers%';
      query_name      | is_active
-----+-----
 findBostonCashiers_CUSTOM | t
 findBostonCashiers_OPT   | f
(2 rows)

(2 rows)
```

Caution: Query results for the fields `input_query` and `annotated_query` are truncated after 8192 characters. You can get the full content of both fields in two ways:

- Use the statement [GET DIRECTED QUERY](#).
- Use [EXPORT_CATALOG](#) to export directed queries.

Identifying Active Directed Queries

Multiple directed queries can map to the same input query. The `is_active` column returned by [GET DIRECTED QUERY](#) clarifies which directed queries are active. If multiple directed queries are active for the same input query, the optimizer uses the first one to be created. In that case, you can use [EXPLAIN](#) to identify which directed query is active.

Note: It is good practice to activate only one directed query at a time for a given input query.

In the following example, `GET DIRECTED QUERY` returns with two directed queries that map to the same input query: `findBostonCashiers_OPT`, and `findBostonCashiers_CUSTOM`. Only `findBostonCashiers_CUSTOM` is flagged as active:

```
=> GET DIRECTED QUERY SELECT Employee_first_name, Employee_last_name FROM Emp_Dimension
    WHERE Employee_city='Boston' and Employee_position='Cashier';
-[ RECORD 1 ]----+
query_name      | findBostonCashiers_OPT
is_active       | f
vertica_version | Vertica Analytic Database v7.2.0
comment         | Optimizer-generated directed query
creation_date   | 2015-09-02 09:36:29.395702
annotated_query | SELECT /*+ verbatim */ Emp_Dimension.Employee_first_name AS Employee_first_name,
Emp_Dimension.Employee_last_name AS Employee_last_name
FROM public.Emp_Dimension AS Emp_Dimension/*+projs('public.Emp_Dimension')*/
WHERE (Emp_Dimension.Employee_city = 'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position =
'Cashier'::varchar(7))
-[ RECORD 2 ]----+
query_name      | findBostonCashiers_CUSTOM
is_active       | t
vertica_version | Vertica Analytic Database v7.2.0-20150902
comment         | Custom directed query
creation_date   | 2015-09-02 13:27:38.225568
annotated_query | SELECT Emp_Dimension.Employee_first_name, Emp_Dimension.Employee_last_name FROM
public.Emp_Dimension/*+Projs('public.Emp_Dimension_Unseg')*/ WHERE ((Emp_Dimension.Employee_city =
'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position = 'Cashier'::varchar(7)))
```

If you run `EXPLAIN` on the same input query, it returns with a query plan that confirms use of `findBostonCashiers_CUSTOM` as the active directed query:

```
=> EXPLAIN SELECT Employee_first_name, Employee_last_name FROM Emp_Dimension WHERE Employee_
city='Boston' AND Employee_position='Cashier';
```

QUERY PLAN

QUERY PLAN DESCRIPTION:

```
EXPLAIN SELECT Employee_first_name, Employee_last_name FROM Emp_Dimension WHERE Employee_
city='Boston' AND Employee_position='Cashier';
```

The following active directed query(query name: findBostonCashiers_CUSTOM) is being executed:

```
SELECT Emp_Dimension.Employee_first_name, Emp_Dimension.Employee_last_name FROM public.Emp_
Dimension/*+Projs('public.Emp_Dimension_Unseg')*/ WHERE ((Emp_Dimension.Employee_city =
'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position = 'Cashier'::varchar(7)))
```

Access Path:

```
+--STORAGE ACCESS for Emp_Dimension [Cost: 154, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.Emp_Dimension_Unseg
| Materialize: Emp_Dimension.Employee_first_name, Emp_Dimension.Employee_last_name
| Filter: (Emp_Dimension.Employee_city = 'Boston')
| Filter: (Emp_Dimension.Employee_position = 'Cashier')
| Execute on: Query Initiator
```

Activating and Deactivating Directed Queries

The optimizer uses only directed queries that are active. If multiple directed queries share the same input query, the optimizer uses the first one to be created.

You activate and deactivate directed queries with [ACTIVATE DIRECTED QUERY](#) and [DEACTIVATE DIRECTED QUERY](#), respectively. For example, the following [ACTIVATE DIRECTED QUERY](#) statement deactivates `findBostonCashiers_OPT` and activates `findBostonCashiers_CUSTOM`:

```
=> DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
DEACTIVATE DIRECTED QUERY;
=> ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
ACTIVATE DIRECTED QUERY;
```

Vertica uses the active directed query for a given query across all sessions until it is explicitly deactivated by `DEACTIVATE DIRECTED QUERY` or removed from storage by [DROP DIRECTED QUERY](#). If a directed query is active at the time of database shutdown, Vertica automatically reactivates it when you restart the database.

After a direct query is deactivated, the query optimizer handles subsequent invocations of the input query by using another directed query, if one is available. Otherwise, it generates its own query plan.

Exporting Directed Queries from the Catalog

Tip: You can also export query plans as directed queries to an external SQL file. See [Batch Query Plan Export](#).

Before upgrading to a new version of Vertica, you can export directed queries for those queries whose optimal performance is critical to your system:

1. Use `EXPORT_CATALOG` with the argument `DIRECTED_QUERIES` to export from the database catalog all current directed queries and their current activation status:

```
=> SELECT EXPORT_CATALOG('../..../export_directedqueries', 'DIRECTED_QUERIES');
EXPORT_CATALOG
-----
Catalog data exported successfully
```

2. `EXPORT_CATALOG` creates a script to recreate the directed queries, as in the following example:

```
SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
FROM public.employee_dimension WHERE ((employee_dimension.employee_city = 'Boston'::varchar(6))
AND (employee_dimension.job_title = 'Cashier'::varchar(7)));
CREATE DIRECTED QUERY CUSTOM findBostonCashiers_OPT COMMENT 'Optimizer-generated directed query'
OPTVER 'Vertica Analytic Database v7.2.3-20160425' PSDATE '2016-04-25 08:17:26.913339' SELECT
/*+ verbatim */ employee_dimension.employee_first_name AS employee_first_name, employee_
dimension.employee_last_name AS employee_last_name
FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/
WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_
title = 'Cashier'::varchar(7));
DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;

SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
FROM public.employee_dimension WHERE ((employee_dimension.employee_city = 'Boston'::varchar(6))
AND (employee_dimension.job_title = 'Cashier'::varchar(7)));
CREATE DIRECTED QUERY CUSTOM findBostonCashiers_CUSTOM COMMENT 'Custom directed query' OPTVER
'Vertica Analytic Database v7.2.3-20160425' PSDATE '2016-04-25 09:15:11.464417' SELECT employee_
dimension.employee_first_name, employee_dimension.employee_last_name FROM public.employee_
dimension/*+Projs('public.Emp_Dimension_Unseg')*/ WHERE ((employee_dimension.employee_city =
'Boston'::varchar(6)) AND (employee_dimension.job_title = 'Cashier'::varchar(7)));
ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
```

Note: The script that `EXPORT_CATALOG` creates specifies to recreate all directed queries with `CREATE DIRECTED QUERY CUSTOM`, regardless of how they were created originally.

3. After the upgrade is complete, remove each directed query from the database catalog with **DROP DIRECTED QUERY**. Alternatively, edit the export script and insert a **DROP DIRECTED QUERY** statement before each **CREATE DIRECTED QUERY** statement. For example, you might modify the script generated earlier with the changes shown in bold:

```
SAVE QUERY SELECT employee_dimension.employee_first_name, ...
DROP DIRECTED QUERY findBostonCashiers_OPT
CREATE DIRECTED QUERY CUSTOM findBostonCashiers_OPT COMMENT 'Optimizer-generated ...
DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;

SAVE QUERY SELECT employee_dimension.employee_first_name, ...
DROP DIRECTED QUERY findBostonCashiers_CUSTOM
CREATE DIRECTED QUERY CUSTOM findBostonCashiers_CUSTOM COMMENT 'Custom directed query'...
ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
```

4. When you run this script, Vertica recreates the directed queries and restores their activation status:

```
=> \i /home/dbadmin/export_directedqueries
SAVE QUERY
DROP DIRECTED QUERY
CREATE DIRECTED QUERY
DEACTIVATE DIRECTED QUERY
SAVE QUERY
DROP DIRECTED QUERY
CREATE DIRECTED QUERY
DEACTIVATE DIRECTED QUERY
```

Dropping Directed Queries

DROP DIRECTED QUERY removes the specified directed query from the database catalog. If the directed query is active, Vertica deactivates it before removal.

For example:

```
=> DROP DIRECTED QUERY findBostonCashiers_CUSTOM;
DROP DIRECTED QUERY
```

Batch Query Plan Export

Before upgrading to a new Vertica version, you might wish to use directed queries to save query plans for possible reuse in the new database. You cannot predict which query plans are likely candidates for reuse, so you probably want to save query plans for many, or all, database queries. However, you run hundreds of queries each day. Saving query plans for each one to

the database catalog through repetitive calls to [CREATE DIRECTED QUERY](#) is impractical. Moreover, doing so can significantly increase catalog size and possibly impact performance.

In this case, you can bypass the database catalog and batch export query plans as directed queries to an external SQL file. By offloading query plan storage, you can save any number of query plans from the current database without impacting catalog size and performance. After the upgrade, you can decide which query plans you wish to retain in the new database, and selectively import the corresponding directed queries.

Vertica provides a set of meta-functions that support this approach:

- [EXPORT_DIRECTED_QUERIES](#) generates query plans from a set of input queries, and writes SQL for creating directed queries that encapsulate those plans.
- [IMPORT_DIRECTED_QUERIES](#) imports to the database catalog directed queries from a SQL file that was generated by [EXPORT_DIRECTED_QUERIES](#).

Exporting Directed Queries

You can batch export any number of query plans as directed queries to an external SQL file, as follows:

1. Create a SQL file that contains the input queries whose query plans you wish to save. See [Output File](#) below.
2. Call the meta-function [EXPORT_DIRECTED_QUERIES](#) on that SQL file. The meta-function takes two arguments:
 - The input queries file.
 - The name of an external file. [EXPORT_DIRECTED_QUERIES](#) writes SQL for creating directed queries to this file. If you supply an empty string, Vertica writes the SQL to standard output. For details, see [Output File](#) below.

For example, the following [EXPORT_DIRECTED_QUERIES](#) statement specifies input file `inputQueries` and output file `outputQueries`:

```
=> SELECT EXPORT_DIRECTED_QUERIES('/home/dbadmin/inputQueries', '/home/dbadmin/outputQueries');
          EXPORT_DIRECTED_QUERIES
-----
 1 queries successfully exported.
Queries exported to /home/dbadmin/outputQueries.

(1 row)
```

Input File

The input file that you supply to `EXPORT_DIRECTED_QUERIES` contains one or more input queries. For each input query, you can optionally specify two fields that are used in the generated directed query:

- `DirQueryName` provides the directed query's unique identifier, a string that conforms to conventions described in [Identifiers](#).
- `DirQueryComment` specifies a quote-delimited string, up to 128 characters.

You format each input query as follows:

```
--DirQueryName=query-name  
--DirQueryComment='comment'  
input-query
```

For example, a file can specify one input query as follows:

```
--DirQueryName=FindEmployeesBoston  
--DirQueryComment='This query finds all Boston employees, ordered by position'  
SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name, employee_  
dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city =  
'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;
```

Output File

`EXPORT_DIRECTED_QUERIES` generates SQL for creating directed queries, and writes the SQL to the specified file or to standard output. In both cases, output conforms to the following format:

```
/* Query: directed-query-name */  
/* Comment: directed-query-comment */  
SAVE QUERY input-query;  
CREATE DIRECTED QUERY CUSTOM 'directed-query-name'  
COMMENT 'directed-query-comment'  
OPTVER 'vertica-release-num'  
PSDATE 'timestamp'  
annotated-query
```

For example, given the previous input, Vertica writes the following output to `/home/dbadmin/outputQueries`:

```
/* Query: FindEmployeesBoston */  
/* Comment: This query finds all Boston employees, ordered by position */  
SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name,  
employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city =
```

```
'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;
CREATE DIRECTED QUERY CUSTOM 'FindEmployeesBoston'
COMMENT 'This query finds all Boston employees, ordered by position'
OPTVER 'Vertica Analytic Database v8.0.1-20161013'
PSDATE '2016-10-13 08:59:58.054505'
SELECT /*+verbatim*/employee_dimension.employee_first_name AS employee_first_name, employee_
dimension.employee_last_name AS employee_last_name, employee_dimension.job_title AS job_title
FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/
WHERE (employee_dimension.employee_city = 'Boston'::varchar(6))
ORDER BY 3 ASC;
```

If a given input query omits `DirQueryName` and `DirQueryComment` fields, `EXPORT_DIRECTED_QUERIES` automatically generates the following output:

- `/* Query: Autaname:timestamp.n */`, where `n` is a zero-based integer index that ensures uniqueness among auto-generated names with the same timestamp.
- `/* Comment: Optimizer-generated directed query */`

For example, the following input file contains one `SELECT` statement, and omits the `DirQueryName` and `DirQueryComment` fields:

```
SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston'::varchar(6))
ORDER BY employee_dimension.job_title
```

Given this file, `EXPORT_DIRECTED_QUERIES` generates the following output :

```
/* Query: Autaname:2016-10-13 09:44:33.527548.0 */
/* Comment: Optimizer-generated directed query */
SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name,
employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city =
'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;
CREATE DIRECTED QUERY CUSTOM 'Autaname:2016-10-13 09:44:33.527548.0'
COMMENT 'Optimizer-generated directed query'
...
```

Error File

If any errors or warnings occur during `EXPORT_DIRECTED_QUERIES` execution, it returns with a message like this one:

```
1 queries successfully exported.
1 warning message was generated.
Queries exported to /home/dbadmin/outputQueries.
See error report, /home/dbadmin/outputQueries.err for details.
```

`EXPORT_DIRECTED_QUERIES` writes all errors and warnings to a file that it creates on the same path as the output file, and uses the output file's base name.

In the previous example, the output filename is `/home/dbadmin/outputQueries`, so `EXPORT_DIRECTED_QUERIES` writes errors to `/home/dbadmin/outputQueries.err`.

The error file can capture a number of errors, such as all instances where `EXPORT_DIRECTED_QUERIES` was unable to create a directed query. In the following example, the error file contains a warning that no name field was supplied for the specified input query, and records the name that was auto-generated for it:

```
-----  
WARNING: Name field not supplied. Using auto-generated name: 'Autoname:2016-10-13 09:44:33.527548.0'  
Input Query: SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name,  
employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city =  
'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;  
END WARNING
```

Importing Directed Queries

After you determine which exported query plans you wish to use in the current database, you import them with [IMPORT_DIRECTED_QUERIES](#). You supply this function with the name of the export file that you created with `EXPORT_DIRECTED_QUERIES`, and the names of directed queries you wish to import. For example:

```
=> SELECT IMPORT_DIRECTED_QUERIES('/home/dbadmin/outputQueries', 'FindEmployeesBoston');  
IMPORT_DIRECTED_QUERIES  
-----  
1 directed queries successfully imported.  
To activate a query named 'my_query1':  
=>ACTIVATE DIRECTED QUERY 'my_query1';  
  
(1 row)
```

After importing the desired directed queries, you must activate them with [ACTIVATE DIRECTED QUERY](#) before you can use them to create query plans.

Half Join and Cross Join Semantics

The Vertica optimizer uses several keywords in directed queries to recreate cross join and half join subqueries. It also supports an additional set of keywords to express complex cross joins and half joins. You can also use these keywords in queries that you execute directly in `vsq`.

Caution: These keywords do not conform with standard SQL; they are intended for use only by the Vertica optimizer.

For details, see the following topics:

- [Half-Join Subquery Semantics](#)
- [Complex Join Semantics](#)

Half-Join Subquery Semantics

The Vertica optimizer uses several keywords in directed queries to recreate half-join subqueries with certain search operators, such as [ANY](#) or [NOT IN](#):

- [SEMI JOIN](#)
- [NULLAWARE ANTI JOIN](#)
- [SEMIALL JOIN](#)
- [ANTI JOIN](#)

SEMI JOIN

Recreates a query that contains a subquery preceded by an [IN](#), [EXIST](#), or [ANY](#) operator and executes a semi-join.

Input query

```
SELECT product_description FROM product_dimension
WHERE product_dimension.product_key IN (SELECT qty_in_stock from inventory_fact);
```

Query plan

```
QUERY PLAN DESCRIPTION:
-----

explain SELECT product_description FROM product_dimension WHERE product_dimension.product_key IN
(SELECT qty_in_stock from inventory_fact);

Access Path:
+-JOIN HASH [Semi] [Cost: 1K, Rows: 30K] (PATH ID: 1) Outer (FILTER) Inner (RESEGMENT)
|   Join Cond: (product_dimension.product_key = VAL(2))
|   Materialize at Output: product_dimension.product_description
|   Execute on: All Nodes
|   +-- Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
|   |   Projection: public.product_dimension
|   |   Materialize: product_dimension.product_key
|   |   Execute on: All Nodes
|   |   Runtime Filter: (SIP1(HashJoin): product_dimension.product_key)
|   +-- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
|   |   Execute on: All Nodes
|   +----> STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
```

```
| | | Projection: public.inventory_fact_b0  
| | | Materialize: inventory_fact.qty_in_stock  
| | | Execute on: All Nodes
```

Optimizer-generated annotated query

```
SELECT /*+ syntactic_join */ product_dimension.product_description AS product_description  
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/  
SEMI JOIN /*+Distrib(F,R),JType(H)*/ (SELECT inventory_fact.qty_in_stock AS qty_in_stock  
FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1  
ON (product_dimension.product_key = subQ_1.qty_in_stock))
```

NULLAWARE ANTI JOIN

Recreates a query that contains a subquery preceded by a **NOT IN** or **!=ALL** operator, and executes a null-aware anti-join.

Input query

```
SELECT product_description FROM product_dimension  
WHERE product_dimension.product_key NOT IN (SELECT qty_in_stock from inventory_fact);
```

Query plan

```
QUERY PLAN DESCRIPTION:  
-----  
  
EXPLAIN SELECT product_description FROM product_dimension WHERE product_dimension.product_key not IN  
(SELECT qty_in_stock from inventory_fact);  
  
Access Path:  
+-JOIN HASH [Anti][NotInAnti] [Cost: 7K, Rows: 30K] (PATH ID: 1) Inner (BROADCAST)  
| Join Cond: (product_dimension.product_key = VAL(2))  
| Materialize at Output: product_dimension.product_description  
| Execute on: Query Initiator  
| +-- Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)  
| | Projection: public.product_dimension_DBD_2_rep_VMartDesign  
| | Materialize: product_dimension.product_key  
| | Execute on: Query Initiator  
| +-- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)  
| | Execute on: All Nodes  
| | +---> STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)  
| | | Projection: public.inventory_fact_DBD_9_seg_VMartDesign_b0  
| | | Materialize: inventory_fact.qty_in_stock  
| | | Execute on: All Nodes
```

Optimizer-generated annotated query

```
SELECT /*+ syntactic_join */ product_dimension.product_description AS product_description
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/
NULLAWARE ANTI JOIN /*+Distrib(L,B),JType(H)*/ (SELECT inventory_fact.qty_in_stock AS qty_in_stock
FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1
ON (product_dimension.product_key = subQ_1.qty_in_stock))
```

SEMIALL JOIN

Recreates a query that contains a subquery preceded by an [ALL](#) operator, and executes a semi-all join.

Input query

```
SELECT product_key, product_description FROM product_dimension
WHERE product_dimension.product_key > ALL (SELECT product_key from inventory_fact);
```

Query plan

```
QUERY PLAN DESCRIPTION:
-----

explain SELECT product_key, product_description FROM product_dimension WHERE product_
dimension.product_key > ALL (SELECT product_key from inventory_fact);

Access Path:
+-JOIN HASH [Semi][All] [Cost: 7M, Rows: 30K] (PATH ID: 1) Outer (FILTER) Inner (BROADCAST)
|  Join Filter: (product_dimension.product_key > VAL(2))
|  Materialize at Output: product_dimension.product_description
|  Execute on: All Nodes
|  +- Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
|  |  Projection: public.product_dimension
|  |  Materialize: product_dimension.product_key
|  |  Execute on: All Nodes
|  +- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
|  |  Execute on: All Nodes
|  |  +---> STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
|  |  |  Projection: public.inventory_fact_b0
|  |  |  Materialize: inventory_fact.product_key
|  |  |  Execute on: All Nodes
```

Optimizer-generated annotated query

```
SELECT /*+ syntactic_join */ product_dimension.product_key AS product_key, product_dimension.product_
description AS product_description
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/
SEMIALL JOIN /*+Distrib(F,B),JType(H)*/ (SELECT inventory_fact.product_key AS product_key FROM
public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1
ON (product_dimension.product_key > subQ_1.product_key))
```

ANTI JOIN

Recreates a query that contains a subquery preceded by a **NOT EXISTS** operator, and executes an anti-join.

Input query

```
SELECT product_key, product_description FROM product_dimension
WHERE NOT EXISTS (SELECT inventory_fact.product_key FROM inventory_fact
WHERE inventory_fact.product_key=product_dimension.product_key);
```

Query plan

QUERY PLAN DESCRIPTION:

```
-----

explain SELECT product_key, product_description FROM product_dimension WHERE NOT EXISTS (SELECT
inventory_fact.product_
key FROM inventory_fact WHERE inventory_fact.product_key=product_dimension.product_key);
```

Access Path:

```
+--JOIN HASH [Anti] [Cost: 703, Rows: 30K] (PATH ID: 1) Outer (FILTER)
|   Join Cond: (VAL(1) = product_dimension.product_key)
|   Materialize at Output: product_dimension.product_description
|   Execute on: All Nodes
| +-- Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
| |   Projection: public.product_dimension_DBD_2_rep_VMartDesign
| |   Materialize: product_dimension.product_key
| |   Execute on: All Nodes
| +-- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
| |   Execute on: All Nodes
| | +----> STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
| | |   Projection: public.inventory_fact_DBD_9_seg_VMartDesign_b0
| | |   Materialize: inventory_fact.product_key
| | |   Execute on: All Nodes
```

Optimizer-generated annotated query

```
SELECT /*+ syntactic_join */ product_dimension.product_key AS product_key, product_dimension.product_
description AS product_description
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/
ANTI JOIN /*+Distrib(F,L),JType(H)*/ (SELECT inventory_fact.product_key AS "inventory_fact.product_
key"
FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1
ON (subQ_1."inventory_fact.product_key" = product_dimension.product_key))
```

Complex Join Semantics

The Vertica optimizer uses a set of keywords to express **complex cross joins** and **half joins**. All complex joins are indicated by the keyword **COMPLEX**, which is inserted before the keyword

JOIN—for example, `CROSS COMPLEX JOIN`. Semantics for complex half joins have an additional requirement, which is detailed [below](#).

Complex Cross Join

Vertica uses the keyword phrase `CROSS COMPLEX JOIN` to describe all complex cross joins. For example:

Input query

```
SELECT
  (SELECT max(sales_quantity) FROM store.store_sales_fact) *
  (SELECT max(sales_quantity) FROM online_sales.online_sales_fact);
```

Query plan

```
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT
  (SELECT max(sales_quantity) FROM store.store_sales_fact) *
  (SELECT max(sales_quantity) FROM online_sales.online_sales_fact);

Access Path:
+-JOIN (CROSS JOIN) [Cost: 4K, Rows: 1 (NO STATISTICS)] (PATH ID: 1)
|   Execute on: Query Initiator
| +- Outer -> JOIN (CROSS JOIN) [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 2)
| |   Execute on: Query Initiator
| | +- Outer -> STORAGE ACCESS for dual [Cost: 10, Rows: 1] (PATH ID: 3)
| | |   Projection: v_catalog.dual_p
| | |   Materialize: dual.dummy
| | |   Execute on: Query Initiator
| | +- Inner -> SELECT [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 4)
| | |   Execute on: Query Initiator
| | | +-> GROUPBY NOTHING [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 5)
| | | |   Aggregates: max(store_sales_fact.sales_quantity)
| | | |   Execute on: All Nodes
| | | +-> STORAGE ACCESS for store_sales_fact [Cost: 1K, Rows: 5M (NO STATISTICS)] (PATH ID: 6)
| | | |   Projection: store.store_sales_fact_super
| | | |   Materialize: store_sales_fact.sales_quantity
| | | |   Execute on: All Nodes
| +- Inner -> SELECT [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 7)
| |   Execute on: Query Initiator
| | +-> GROUPBY NOTHING [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 8)
| | |   Aggregates: max(online_sales_fact.sales_quantity)
| | |   Execute on: All Nodes
| | +-> STORAGE ACCESS for online_sales_fact [Cost: 1K, Rows: 5M (NO STATISTICS)] (PATH ID: 9)
| | |   Projection: online_sales.online_sales_fact_super
| | |   Materialize: online_sales_fact.sales_quantity
| | |   Execute on: All Nodes
```

Optimizer-generated annotated query

The following annotated query returns the same results as the input query shown earlier. As with all optimizer-generated annotated queries, you can execute this query directly in vsql, either as written or with modifications:

```
SELECT /*+syntactic_join,verbatim*/ (subQ_1.max * subQ_2.max) AS "?column?"
FROM ((v_catalog.dual AS dual CROSS COMPLEX JOIN /*+Distrib(L,L),JType(H)*/
(SELECT max(store_sales_fact.sales_quantity) AS max
FROM store_sales_fact AS store_sales_fact/*+projs('store_sales_fact')*/) AS subQ_1 )
CROSS COMPLEX JOIN /*+Distrib(L,L),JType(H)*/ (SELECT max(online_sales_fact.sales_quantity) AS max
FROM online_sales_fact AS online_sales_fact/*+projs('online_sales_fact')*/)
AS subQ_2 )
```

Complex Half Join

Complex half joins are expressed by one of the following keywords:

- SEMI COMPLEX JOIN
- NULLAWARE ANTI COMPLEX JOIN
- SEMIALL COMPLEX JOIN
- ANTI COMPLEX JOIN

An additional requirement applies to all complex half joins: each subquery's SELECT list ends with a dummy column (labeled as false) that invokes the Vertica meta-function `complex_join_marker()`. As the subquery processes each row, `complex_join_marker()` returns true or false to indicate the row's inclusion or exclusion from the result set. The result set returns with this flag to the outer query, which can use the flag from this and other subqueries to filter its own result set.

For example, the query optimizer rewrites the following input query as a NULLAWARE ANTI COMPLEX JOIN. The join returns all rows from the subquery with their `complex_join_marker()` flag set to the appropriate Boolean value.

Input query

```
SELECT product_dimension.product_description FROM public.product_dimension
WHERE (NOT (product_dimension.product_key NOT IN (SELECT inventory_fact.qty_in_stock FROM
public.inventory_fact)));
```

Query plan

QUERY PLAN DESCRIPTION:

```
EXPLAIN SELECT product_dimension.product_description FROM public.product_dimension
WHERE (NOT (product_dimension.product_key NOT IN (SELECT inventory_fact.qty_in_stock FROM
```

```
public.inventory_fact));  
  
Access Path:  
+-JOIN HASH [Anti][NotInAnti] [Cost: 3K, Rows: 30K] (PATH ID: 1) Inner (BROADCAST)  
| Join Cond: (product_dimension.product_key = VAL(2))  
| Materialize at Output: product_dimension.product_description  
| Filter: (NOT VAL(2))  
| Execute on: All Nodes  
| +- Outer -> STORAGE ACCESS for product_dimension [Cost: 56, Rows: 60K] (PATH ID: 2)  
| | Projection: public.product_dimension_super  
| | Materialize: product_dimension.product_key  
| | Execute on: All Nodes  
| +- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)  
| | Execute on: All Nodes  
| | +---> STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)  
| | | Projection: public.inventory_fact_super  
| | | Materialize: inventory_fact.qty_in_stock  
| | | Execute on: All Nodes
```

Optimizer-generated annotated query

The following annotated query returns the same results as the input query shown earlier. As with all optimizer-generated annotated queries, you can execute this query directly in vsql, either as written or with modifications. For example, you can control the outer query's output by modifying how its predicate evaluates the flag `subQ_1."false"`.

```
SELECT /*+syntactic_join,verbatim*/ product_dimension.product_description AS product_description  
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/  
NULLAWARE ANTI COMPLEX JOIN /*+Distrib(L,B),JType(H)*/  
  (SELECT inventory_fact.qty_in_stock AS qty_in_stock, complex_join_marker() AS "false"  
   FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1  
ON (product_dimension.product_key = subQ_1.qty_in_stock)) WHERE (NOT subQ_1."false")
```

Restrictions

Directed queries support a wide range of queries; however, a number of exceptions apply. Vertica handles all exceptions through optimizer-generated warnings. The sections below divide these restrictions into several categories.

Tables and Projections

The following restrictions apply:

- Optimizer-generated directed queries do not support queries that reference system tables or Data Collector tables. One exception applies: explicit and implicit references to [V_CATALOG.DUAL](#).

- Optimizer-generated directed queries do not support queries that include tables with access policies.
- Directed queries do not support tables without projections.

Functions

Queries are not supported that include the following functions:

- [Vertica meta-functions](#)
- [Pattern-matching functions](#)
- `GROUPING_ID` with no arguments

Operators and Clauses

Queries are not supported that include the following:

- WITH clauses [when materialization is enabled](#).
- Queries that include date/time literals that reference the current time, such as NOW or YESTERDAY

Data Types

Queries are not supported that include [GEOMETRY data types](#).

Using Text Search

Text Search allows you to quickly search the contents of a single CHAR, VARCHAR, LONG VARCHAR, VARBINARY, or LONG VARBINARY field within a table to locate a specific keyword.

You can use this feature on columns that are queried repeatedly regarding their contents. After you create the text index, DML operations become slightly slower on the source table. This performance change results from syncing the text index and source table. Any time an operation is performed on the source table, the text index updates in the background. Regular queries on the source table are not affected.

The text index contains all of the words from the source table's text field, as well as any other additional columns you included during index creation. Additional columns are not indexed, their values are just passed through to the text index. The text index is like any other table in the Vertica Analytics Platform, except it is linked to the source table internally.

You first create a text index on the table you plan to search. Then, after you have indexed your table, you can run a query against the text index for a specific keyword. This query returns a doc_id for each instance of the keyword. After querying the text index, joining the text index back to the source table should give a significant performance improvement over directly querying the source table about the contents of its text field.

Important: Do not alter the contents or definitions of the text index. If the contents or definitions of the text index are altered, then the results do not appropriately match the source table.

Creating a Text Index

In the following example, you perform a text search using a source table called t_log. This source table has two columns:

- One column containing the table's primary key
- Another column containing log file information

You must associate a projection with the source table. Use a projection that is sorted by the primary key and either segmented by hash(id) or unsegmented. You can define this projection on the source table, along with any other existing projections.

Create a text index on the table for which you want to perform a text search.

```
=> CREATE TEXT INDEX text_index ON t_log (id, text);
```

The text index contains two columns:

- `doc_id` uses the unique identifier from the source table.
- `token` is populated with text strings from the designated column from the source table. The word column results from tokenizing and stemming the words found in the text column.

If your table is partitioned then your text index also contains a third column named *partition*.

```
=> SELECT * FROM text_index;
      token      | doc_id | partition
-----+-----+-----
<info>          |      6 |      2014
<warning>      |      2 |      2014
<warning>      |      3 |      2014
<warning>      |      4 |      2014
<warning>      |      5 |      2014
database        |      6 |      2014
execute:        |      6 |      2014
object          |      4 |      2014
object          |      5 |      2014
[catalog]       |      4 |      2014
[catalog]       |      5 |      2014
```

You create a text index on a source table only once. In the future, you do not have to re-create the text index each time the source table is updated or changed.

Your text index stays synchronized to the contents of the source table through any operation that is run on the source table. These operations include, but are not limited to:

- COPY
- INSERT
- UPDATE
- DELETE
- DROP PARTITION
- MOVE_PARTITION_TO_TABLE
When you move or swap partitions in a source table that is indexed, verify that the destination table already exists and is indexed in the same way.

Creating a Text Index on a Flex Table

In the following example, you create a text index on a flex table. The example assumes that you have created a flex table called `mountains`. See [Getting Started](#) in Using Flex Tables to create the flex table used in this example.

Before you can create a text index on your flex table, add a primary key constraint to the flex table.

```
=> ALTER TABLE mountains ADD PRIMARY KEY (__identity__);
```

Create a text index on the table for which you want to perform a text search. Tokenize the `__raw__` column with the `FlexTokenizer` and specify the data type as `LONG VARBINARY`. It is important to use the `FlexTokenizer` when creating text indices on flex tables because the data type of the `__raw__` column differs from the default `StringTokenizer`.

```
=> CREATE TEXT INDEX flex_text_index ON mountains(__identity__, __raw__) TOKENIZER  
public.FlexTokenizer(long varbinary);
```

The text index contains two columns:

- `doc_id` uses the unique identifier from the source table.
- `token` is populated with text strings from the designated column from the source table. The word column results from tokenizing and stemming the words found in the text column.

If your table is partitioned then your text index also contains a third column named *partition*.

```
=> SELECT * FROM flex_text_index;  
  token | doc_id  
-----+-----  
  50.6  |      5  
  Mt    |      5  
  Washington |      5  
  mountain |      5  
  12.2  |      3  
  15.4  |      2  
  17000 |      3  
  29029 |      2  
  Denali |      3  
  Helen |      2  
  Mt    |      2  
  St    |      2  
  mountain |      3  
  volcano |      2  
  29029 |      1  
  34.1  |      1  
  Everest |      1
```

```
mountain | 1
14000   | 4
Kilimanjaro | 4
mountain | 4
(21 rows)
```

You create a text index on a source table only once. In the future, you do not have to re-create the text index each time the source table is updated or changed.

Your text index stays synchronized to the contents of the source table through any operation that is run on the source table. These operations include, but are not limited to:

- COPY
- INSERT
- UPDATE
- DELETE
- DROP PARTITION

- MOVE_PARTITION_TO_TABLE

When you move or swap partitions in a source table that is indexed, verify that the destination table already exists and is indexed in the same way.

Searching a Text Index

After you create a text index, write a query to run against the index to search for a specific keyword.

In the following example, you use a WHERE clause to search for the keyword <WARNING> in the text index. The WHERE clause should use the stemmer you used to create the text index. When you use the STEMMER keyword, it stems the keyword to match the keywords in your text index. If you did not use the STEMMER keyword, then the default stemmer is `v_txtindex.StemmerCaseInsensitive`. If you used STEMMER NONE, then you can omit STEMMER keyword from the WHERE clause.

```
=> SELECT * FROM text_index WHERE token = v_txtindex.StemmerCaseInsensitive('<WARNING>');
 token | doc_id
-----+-----
<warning> | 2
<warning> | 3
<warning> | 4
<warning> | 5
```

(4 rows)

Next, write a query to display the full contents of the source table that match the keyword you searched for in the text index.

```
=> SELECT * FROM t_log WHERE id IN (SELECT doc_id FROM text_index WHERE token = v_
txtindex.StemmerCaseInsensitive('<WARNING>'));
id | date | text
-----+-----+-----
-----
4 | 2014-06-04 | 11:00:49.568 unknown:0x7f9207607700 [Catalog] <WARNING> validateDependencies:
Object 45035968
5 | 2014-06-04 | 11:00:49.568 unknown:0x7f9207607700 [Catalog] <WARNING> validateDependencies:
Object 45030
2 | 2013-06-04 | 11:00:49.568 unknown:0x7f9207607700 [Catalog] <WARNING> validateDependencies:
Object 4503
3 | 2013-06-04 | 11:00:49.568 unknown:0x7f9207607700 [Catalog] <WARNING> validateDependencies:
Object 45066
(4 rows)
```

Use the `doc_id` to find the exact location of the keyword in the source table. The `doc_id` matches the unique identifier from the source table. This matching allows you to quickly find the instance of the keyword in your table.

Performing a Case-Sensitive and Case-Insensitive Text Search Query

Your text index is optimized to match all instances of words depending upon your stemmer. By default, the case insensitive stemmer is applied to all text indices that do not specify a stemmer. Therefore, if the queries you plan to write against your text index are case sensitive, then Micro Focus recommends you use a case sensitive stemmer to build your text index.

The following examples show queries that match case-sensitive and case-insensitive words that you can use when performing a text search.

This query finds case-insensitive records in a case insensitive text index:

```
=> SELECT * FROM t_log WHERE id IN (SELECT doc_id FROM text_index WHERE token = v_
txtindex.StemmerCaseInsensitive('warning'));
```

This query finds case-sensitive records in a case sensitive text index:

```
=> SELECT * FROM t_log_case_sensitive WHERE id IN (SELECT doc_id FROM text_index WHERE token = v_
txtindex.StemmerCaseSensitive('Warning'));
```

Including and Excluding Keywords in a Text Search Query

Your text index also allows you to perform more detailed queries to find multiple keywords or omit results with other keywords. The following example shows a more detailed query that you can use when performing a text search.

In this example, `t_log` is the source table, and `text_index` is the text index. The query finds records that either contain:

- Both the words '<WARNING>' and 'validate'
- Only the word '[Log]' and does not contain 'validateDependencies'

```
SELECT * FROM t_log where (  
  id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive('<WARNING>'))  
  AND ( id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive  
( 'validate'  
  OR id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive(''  
[Log]'))  
  AND NOT (id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive  
( 'validateDependencies'))));
```

This query returns the following results:

```
id | date | | text  
-----+-----+-----+-----  
-----  
11 | 2014-05-04 | 11:00:49.568 | unknown:0x7f9207607702 [Log] <WARNING> validate: Object 4503 via fld  
num_all_roles  
13 | 2014-05-04 | 11:00:49.568 | unknown:0x7f9207607706 [Log] <WARNING> validate: Object 45035 refers  
to root_i3  
14 | 2014-05-04 | 11:00:49.568 | unknown:0x7f9207607708 [Log] <WARNING> validate: Object 4503 refers  
to int_2  
17 | 2014-05-04 | 11:00:49.568 | unknown:0x7f9207607700 [Txn] <WARNING> Begin validate Txn: fff0ed17  
catalog editor  
(4 rows)
```

Dropping a Text Index

Dropping a text index removes the specified text index from the database.

You can drop a text index when:

- It is no longer queried frequently.
- An administrative task needs to be performed on the source table and requires the text index to be dropped.

Dropping the text index does not drop the source table associated with the text index. However, if you drop the source table associated with a text index, then that text index is also dropped. Vertica considers the text index a dependent object.

The following example illustrates how to drop a text index named `text_index`:

```
=> DROP TEXT INDEX text_index;  
DROP INDEX
```

Stemmers and Tokenizers

Vertica provides default stemmers and tokenizers. You can also create your own custom stemmers and tokenizers. The following topics explain the default stemmers and tokenizers, and the requirements for creating custom stemmers and tokenizers in Vertica.

- [Vertica Stemmers](#)
- [Vertica Tokenizers](#)
- [Configuring a Tokenizer](#)
- [Requirements for Custom Stemmers and Tokenizers](#)

Vertica Stemmers

Vertica *stemmers* use the Porter stemming algorithm to find words derived from the same base/root word. For example, if you perform a search on a text index for the keyword *database*, you might also want to get results containing the word *databases*.

To achieve this type of matching, Vertica stores words in their stemmed form when using any of the `v_txtindex` stemmers.

The Vertica Analytics Platform provides the following stemmers:

Name	Description
<code>v_txtindex.Stemmer(long varchar)</code>	Not sensitive to case; outputs lowercase words.

Name	Description
	Stems strings from a Vertica table. Alias of StemmerCaseInsensitive.
v_txtindex.StemmerCaseSensitive(long varchar)	Sensitive to case. Stems strings from a Vertica table.
v_txtindex.StemmerCaseInsensitive (long varchar)	Default stemmer used if no stemmer is specified when creating a text index. Not sensitive to case; outputs lowercase words. Stems strings from a Vertica table.
v_txtindex.caseInsensitiveNoStemming (long varchar)	Not sensitive to case; outputs lowercase words. Does not use the Porter Stemming algorithm.

Examples

The following examples show how to use a stemmer when creating a text index.

Create a text index using the StemmerCaseInsensitive stemmer:

```
=> CREATE TEXT INDEX idx_100 ON top_100 (id, feedback) STEMMER v_txtindex.StemmerCaseInsensitive(long varchar)
                                     TOKENIZER v_txtindex.StringTokenizer
(long varchar);
```

Create a text index using the StemmerCaseSensitive stemmer:

```
=> CREATE TEXT INDEX idx_unstruc ON unstruc_data (__identity__, __raw__) STEMMER v_
txtindex.StemmerCaseSensitive(long varchar)
                                     TOKENIZER
public.FlexTokenizer(long varbinary);
```

Create a text index without using a stemmer:

```
=> CREATE TEXT INDEX idx_logs FROM sys_logs ON (id, message) STEMMER NONE TOKENIZER v_
txtindex.StringTokenizer(long varchar);
```

Vertica Tokenizers

The Vertica Analytics Platform provides the following pre-configured tokenizers:

Name	Description
public.FlexTokenizer(long varbinary)	Splits the values in your Flex Table by white space.
v_txtindex.StringTokenizer(long varchar)	Splits the string into words by splitting on white space.
v_txtindex.AdvancedLogTokenizer	Uses the default parameters for all tokenizer parameters. For more information, see Advanced Log Tokenizer .
v_txtindex.BasicLogTokenizer	Uses the default values for all tokenizer parameters except minorseparator, which is set to an empty list. For more information, see Basic Log Tokenizer .
v_txtindex.WhitespaceLogTokenizer	Uses default values for tokenizer parameters, except for majorseparators, which uses E' \t\n\f\r'; and minorseparator, which uses an empty list. For more information, see Whitespace Log Tokenizer .

Vertica also provides the following tokenizer, which is not pre-configured:

Name	Description
v_txtindex.ICUTokenizer	Supports multiple languages. Tokenizes based on the conventions of the language you set in the locale parameter. For more information, see ICU Tokenizer.

Examples

The following examples show how you can use a pre-configured tokenizer when creating a text index.

Use the StringTokenizer to create an index from the top_100:

```
=> CREATE TEXT INDEX idx_100 FROM top_100 on (id, feedback)
      TOKENIZER v_txtindex.StringTokenizer(long varchar)
      STEMMER v_txtindex.StemmerCaseInsensitive(long varchar);
```

Use the FlexTokenizer to create an index from unstructured data:

```
=> CREATE TEXT INDEX idx_unstruc FROM unstruc_data on (__identity__, __raw__)
      TOKENIZER public.FlexTokenizer(long varbinary)
      STEMMER v_txtindex.StemmerCaseSensitive(long varchar);
```

Advanced Log Tokenizer

Returns tokens that can include minor separators. You can use this tokenizer in situations when your tokens are separated by whitespace or various punctuation. The advanced log tokenizer offers more granularity than the basic log tokenizer in defining separators through the addition of minor separators. This approach is frequently appropriate for analyzing log files.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using [REBALANCE_CLUSTER](#) before using a Vertica tokenizer.

Parameters

Parameter Name	Parameter Value
stopwordscaseinsensitive	' '
minorseparators	E' / :=@. - \$#% \ _ '
majorseparators	E' [] < > () { } ! ; , ' ' " * & ? + \ r \ n \ t '
minLength	'2'
maxLength	'128'
used	'True'

Examples

The following example shows how you can create a text index, from the table foo, using the Advanced Log Tokenizer without a stemmer.

```
=> CREATE TABLE foo (id INT PRIMARY KEY NOT NULL, text VARCHAR(250));
=> COPY foo FROM STDIN;
End with a backslash and a period on a line by itself.
>> 1|2014-05-10 00:00:05.700433 %ASA-6-302013: Built outbound TCP connection 9986454 for
outside:101.123.123.111/443 (101.123.123.111/443)
>> \.
=> CREATE PROJECTION foo_projection AS SELECT * FROM foo ORDER BY id
                               SEGMENTED BY HASH(id) ALL NODES KSAFE;
=> CREATE TEXT INDEX indexfoo_AdvancedLogTokenizer ON foo (id, text)
                               TOKENIZER v_txtindex.AdvancedLogTokenizer(LONG VARCHAR) STEMMER NONE;
=> SELECT * FROM indexfoo_AdvancedLogTokenizer;
      token                | doc_id
-----+-----
%ASA-6-302013:           |      1
```

```

00 | 1
00:00:05.700433 | 1
05 | 1
10 | 1
101 | 1
101.123.123.111/443 | 1
111 | 1
123 | 1
2014 | 1
2014-05-10 | 1
302013 | 1
443 | 1
700433 | 1
9986454 | 1
ASA | 1
Built | 1
TCP | 1
connection | 1
for | 1
outbound | 1
outside | 1
outside:101.123.123.111/443 | 1
(23 rows)

```

Basic Log Tokenizer

Returns tokens that exclude specified minor separators. You can use this tokenizer in situations when your tokens are separated by whitespace or various punctuation. This approach is frequently appropriate for analyzing log files.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using [REBALANCE_CLUSTER](#) before using a Vertica tokenizer.

Parameters

Parameter Name	Parameter Value
stopwordscaseinsensitive	' '
minorseparators	' '
majorseparators	E' []<>(){} !;, '"*&?+\r\n\t'
minLength	'2'
maxLength	'128'
used	'True'

Examples

The following example shows how you can create a text index, from the table foo, using the Basic Log Tokenizer without a stemmer.

```
=> CREATE TABLE foo (id INT PRIMARY KEY NOT NULL,text VARCHAR(250));
=> COPY foo FROM STDIN;
End with a backslash and a period on a line by itself.
>> 1|2014-05-10 00:00:05.700433 %ASA-6-302013: Built outbound TCP connection 9986454 for
outside:101.123.123.111/443 (101.123.123.111/443)
>> \.
=> CREATE PROJECTION foo_projection AS SELECT * FROM foo ORDER BY id
          SEGMENTED BY HASH(id) ALL NODES KSAFE;
=> CREATE TEXT INDEX indexfoo_BasicLogTokenizer ON foo (id, text)
          TOKENIZER v_txtindex.BasicLogTokenizer(LONG VARCHAR) STEMMER NONE;
=> SELECT * FROM indexfoo_BasicLogTokenizer;
      token          | doc_id
-----+-----
%ASA-6-302013:      |      1
00:00:05.700433    |      1
101.123.123.111/443 |      1
2014-05-10         |      1
9986454            |      1
Built              |      1
TCP                |      1
connection         |      1
for                |      1
outbound           |      1
outside:101.123.123.111/443 |      1
(11 rows)
```

Whitespace Log Tokenizer

Returns only tokens surrounded by whitespace. You can use this tokenizer in situations where you want the tokens in your source document to be separated by whitespace characters only. This approach lets you retain the ability to set stop words and token length limits.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using [REBALANCE_CLUSTER](#) before using a Vertica tokenizer.

Parameters

Parameter Name	Parameter Value
stopwordscaseinsensitive	' '
minorseparators	' '

Parameter Name	Parameter Value
majorseparators	E' \t\n\f\r'
minLength	'2'
maxLength	'128'
used	'True'

Examples

The following example shows how you can create a text index, from the table foo, using the Whitespace Log Tokenizer without a stemmer.

```
=> CREATE TABLE foo (id INT PRIMARY KEY NOT NULL,text VARCHAR(250));
=> COPY foo FROM STDIN;
End with a backslash and a period on a line by itself.
>> 1|2014-05-10 00:00:05.700433 %ASA-6-302013: Built outbound TCP connection 998 6454 for
outside:101.123.123.111/443 (101.123.123.111/443)
>> \.
=> CREATE PROJECTION foo_projection AS SELECT * FROM foo ORDER BY id
                               SEGMENTED BY HASH(id) ALL NODES KSAFE;
=> CREATE TEXT INDEX indexfoo_WhitespaceLogTokenizer ON foo (id, text)
                               TOKENIZER v_txtindex.WhitespaceLogTokenizer(LONG VARCHAR) STEMMER NONE;
=> SELECT * FROM indexfoo_WhitespaceLogTokenizer;
      token          | doc_id
-----+-----
%ASA-6-302013:      |      1
(101.123.123.111/443) |      1
00:00:05.700433    |      1
2014-05-10         |      1
6454               |      1
998               |      1
Built              |      1
TCP               |      1
connection         |      1
for               |      1
outbound          |      1
outside:101.123.123.111/443 |      1
(12 rows)
```

ICU Tokenizer

Supports multiple languages. You can use this tokenizer to identify word boundaries in languages other than English, including Asian languages that are not separated by whitespace.

The ICU Tokenizer is not pre-configured. You configure the tokenizer by first creating a User-Defined Transform Function (UDTF). Then set the parameter, locale, to identify the language to tokenizer.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using [REBALANCE_CLUSTER](#) before using a Vertica tokenizer.

Parameters

Parameter Name	Parameter Value
locale	Uses the POSIX naming convention: language[_COUNTRY] Identify the language using its ISO-639 code, and the country using its ISO-3166 code. For example, the parameter value for simplified Chinese is zh_CN, and the value for Spanish is es_ES. The default value is English if you do not specify a locale.

Example

The following example steps show how you can configure the ICU Tokenizer for simplified Chinese, then create a text index from the table foo, which contains Chinese characters.

For more on how to configure tokenizers, see [Configuring a Tokenizer](#).

1. Create the tokenizer using a UDTF. The example tokenizer is named ICUChineseTokenizer.

```
VMart=> CREATE OR REPLACE TRANSFORM FUNCTION v_txtindex.ICUChineseTokenizer AS LANGUAGE 'C++'  
NAME 'ICUTokenizerFactory' LIBRARY v_txtindex.logSearchLib NOT FENCED;  
CREATE TRANSFORM FUNCTION
```

2. Get the procedure ID of the tokenizer.

```
VMart=> SELECT proc_oid from vs_procedures where procedure_name = 'ICUChineseTokenizer';  
  
   proc_oid  
-----  
  45035996280452894  
(1 row)
```

3. Set the parameter, locale, to simplified Chinese. Identify the tokenizer using its procedure ID.

```
VMart=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('locale','zh_CN' using parameters proc_  
oid='45035996280452894');  
   SET_TOKENIZER_PARAMETER  
-----  
   t
```

```
(1 row)
```

4. Lock the tokenizer.

```
VMart=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('used','true' using parameters proc_
oid='45035996273762696');
SET_TOKENIZER_PARAMETER
-----
t
(1 row)
```

5. Create an example table, foo, containing simplified Chinese text to index.

```
VMart=> CREATE TABLE foo(doc_id integer primary key not null,text varchar(250));
CREATE TABLE

VMart=> INSERT INTO foo values(1, u&'\4E2D\534E\4EBA\6C11\5171\548C\56FD');
OUTPUT
-----
1
```

6. Create an index, index_example, on the table foo. The example creates the index without a stemmer; Vertica stemmers work only on English text. Using a stemmer for English on non-English text can cause incorrect tokenization.

```
VMart=> CREATE TEXT INDEX index_example ON foo (doc_id, text) TOKENIZER v_
txtindex.ICUChineseTokenizer(long varchar) stemmer none;
CREATE INDEX
```

7. View the new index.

```
VMart=> SELECT * FROM index_example ORDER BY token,doc_id;
token | doc_id
-----+-----
中华  |      1
人民  |      1
共和国 |      1
(3 rows)
```

Configuring a Tokenizer

You configure a tokenizer by creating a User-Defined Transform Function (UDTF) using one of the two base UDTFs in the `v_txtindex.AdvTxtSearchLib` library. The library contains two base tokenizers: one for Log Words and one for Ngrams. You can configure each base function with or without positional relevance.

You can choose among several different tokenizer base configurations:

Type	Position	Without Position
Ngram	logNgramTokenizerPositionFactory	logNgramTokenizerFactory
Words	logWordITokenizerPositionFactory	logWordITokenizerFactory

Create a logWord tokenizer without positional relevance:

```
=> CREATE TRANSFORM FUNCTION v_txtindex.fooTokenizer AS LANGUAGE 'C++' NAME  
'logWordITokenizerFactory' LIBRARY v_txtindex.logSearchLib NOT FENCED;
```

Retrieve a Tokenizer's proc_oid

After you create the tokenizer, Vertica writes the name and proc_oid to the system table vs_procedures. You must retrieve the tokenizer's proc_oid to perform additional configuration.

Enter the following query, substituting your own tokenizer name:

```
=> SELECT proc_oid FROM vs_procedures WHERE procedure_name = 'fooTokenizer';
```

Set Tokenizer Parameters

Use the tokenizer's proc_oid to configure the tokenizer. See [Configuring a Tokenizer](#) for more information about getting the proc_oid of your tokenizer. The following examples show how you can configure each of the tokenizer parameters:

Configure stop words:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('stopwordscaseinsensitive', 'for,the' USING PARAMETERS  
proc_oid='45035996274128376');
```

Configure major separators:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('majorseparators', E'{}()&[]' USING PARAMETERS proc_  
oid='45035996274128376');
```

Configure minor separators:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('minorseparators', '-,$' USING PARAMETERS proc_  
oid='45035996274128376');
```

Configure minimum length:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('minlength', '1' USING PARAMETERS proc_  
oid='45035996274128376');
```

Configure maximum length:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('maxlength', '140' USING PARAMETERS proc_
oid='45035996274128376');
```

Configure ngramssize:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('ngramssize', '2' USING PARAMETERS proc_
oid='45035996274128376');
```

Lock Tokenizer Parameters

When you finish configuring the tokenizer, set the parameter, `used`, to `True`. After changing this setting, you are no longer able to alter the parameters of the tokenizer. At this point, the tokenizer is ready for you to use to create a text index.

Configure the `used` parameter:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('used', 'True' USING PARAMETERS proc_
oid='45035996274128376');
```

See Also

[SET_TOKENIZER_PARAMETER](#)

View Tokenizer Parameters

After creating a custom tokenizer, you can view the tokenizer's parameter settings in either of two ways:

- Use the `GET_TOKENIZER_PARAMETER` — [View individual tokenizer parameter settings](#).
- Use the `READ_CONFIG_FILE` — [View all tokenizer parameter settings](#).

View Individual Tokenizer Parameter Settings

If you need to see an individual parameter setting for a tokenizer, you can use `GET_TOKENIZER_PARAMETER` to see specific tokenizer parameter settings:

```
=> SELECT v_txtindex.GET_TOKENIZER_PARAMETER('majorseparators' USING PARAMETERS proc_
oid='45035996274126984');
getTokenizerParameter
-----
{ } ( ) & [ ]
```

(1 row)

For more information, see [GET_TOKENIZER_PARAMETER](#).

View All Tokenizer Parameter Settings

If you need to see all of the parameters for a tokenizer, you can use `READ_CONFIG_FILE` to see all of the parameter settings for your tokenizer:

```
=> SELECT v_txtindex.READ_CONFIG_FILE( USING PARAMETERS proc_oid='45035996274126984') OVER();
      config_key | config_value
-----+-----
majorseparators | {}()&[]
maxlength       | 140
minlength       | 1
minorseparators | -, $
stopwordscaseinsensitive | for, the
                | type | 1
                | used | true
(7 rows)
```

If the parameter, `used`, is set to `False`, then you can only view the parameters that have been applied to the tokenizer.

Note: Vertica automatically supplies the value for `Type`, unless you are using an ngram tokenizer, which allows you to set it.

For more information, see [READ_CONFIG_FILE](#).

Delete Tokenizer Config File

Use the `DELETE_TOKENIZER_CONFIG_FILE` function to delete a tokenizer configuration file. This function does not delete the User- Defined Transform Function (UDTF). It only deletes the configuration file associated with the UDTF.

Delete the tokenizer configuration file when the parameter, `used`, is set to `False`:

```
=> SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE(USING PARAMETERS proc_oid='45035996274127086');
```

Delete the tokenizer configuration file with the parameter, `confirm`, set to `True`. This setting forces the configuration file deletion, even if the parameter, `used`, is also set to `True`:

```
=> SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE(USING PARAMETERS proc_oid='45035996274126984',
confirm='true');
```

For more information, see [DELETE_TOKENIZER_CONFIG_FILE](#).

Requirements for Custom Stemmers and Tokenizers

Sometimes, you may want specific tokenization or stemming behavior that differs from what Vertica provides. In such cases, you can implement your own custom User Defined Extensions (UDx) to replace the stemmer or tokenizer. For more information about building custom UDxs see [Developing User-Defined Extensions \(UDxs\)](#).

Before implementing a custom stemmer or tokenizer in Vertica verify that the UDx extension meets these requirements.

Note: Custom tokenizers can return multi-column text indices.

Vertica Stemmer Requirements

Comply with these requirements when you create custom stemmers:

- Must be a User Defined Scalar Function (UDSF) or a SQL Function
- Can be written in C++, Java, or R
- Volatility set to stable or immutable

Supported Data Input Types:

- Varchar
- Long varchar

Supported Data Output Types:

- Varchar
- Long varchar

Vertica Tokenizer Requirements

To create custom tokenizers, follow these requirements:

- Must be a User Defined Transform Function (UDTF)
- Can be written in C++, Java, or R

- Input type must match the type of the input text

Supported Data Input Types:

- Char
- Varchar
- Long varchar
- Varbinary
- Long varbinary

Supported Data Output Types:

- Varchar
- Long varchar

Bulk-Loading Data

This section describes different methods to bulk-load data into a Vertica database using the `COPY` statement. In its simplest form, `COPY` copies data from a source to a file, as follows:

```
COPY target-table FROM data-source
```

Source data can be a data stream, or a file stored on the host or client. `COPY` supports numerous options, including:

- Incoming data format
- Metadata about the data load
- The best parser to use for different types of data
- Parallel load streams to load data
- Data transforms as `COPY` loads data
- Saving rejected data from parse errors to files or a table

Checking Data Format Before or After Loading

Vertica supports loading data files in the Unicode UTF-8 format. You can load ASCII data, which is UTF-8 compatible. Character sets like ISO 8859-1 (Latin1) are incompatible with UTF-8 and are not supported.

Before loading data from text files, you can use several Linux tools to ensure that your data is in UTF-8 format. The `file` command reports the encoding of any text files. For example:

```
$ file Date_Dimension.tbl
Date_Dimension.tbl: ASCII text
```

The `file` command could indicate ASCII TEXT even though the file contains multibyte characters.

To check for multibyte characters in an ASCII file, use the `wc` command. For example:

```
$ wc Date_Dimension.tbl
1828  5484 221822 Date_Dimension.tbl
```

If the `wc` command returns an error such as `Invalid or incomplete multibyte or wide character`, the data file is using an incompatible character set.

This example describes files that are not UTF-8 data files. Two text files have filenames starting with the string `data`. To check their format, use the `file` command as follows:

```
$ file data*
data1.txt: Little-endian UTF-16 Unicode text
data2.txt: ISO-8859 text
```

The results indicate that neither of the files is in UTF-8 format.

Converting Files Before Loading Data

To convert files before loading them into Vertica, use the `iconv` UNIX command. For example, to convert the `data2.txt` file from the previous example, use the `iconv` command as follows:

```
iconv -f ISO88599 -t utf-8 data2.txt > data2-utf8.txt
```

See the man pages for `file` and `iconv` for more information.

Checking UTF-8 Compliance After Loading Data

After loading data, use the `ISUTF8` function to verify that all of the string-based data in the table is in UTF-8 format. For example, if you loaded data into a table named `nametable` that has a `VARCHAR` column named `name`, you can use this statement to verify that all of the strings are UTF-8 encoded:

```
=> SELECT name FROM nametable WHERE ISUTF8(name) = FALSE;
```

If all of the strings are in UTF-8 format, the query should not return any rows.

Performing the Initial Database Load

To perform the initial database load, use `COPY` with its `DIRECT` parameter from `vsq`.

Tip: Vertica supports multiple schema types. If you have a star schema, load the smaller tables before you load the largest tables.

Only a superuser can use the `COPY` statement to bulk load data. Two exceptions to the superuser requirement are to:

1. Run COPY to load from a stream on the host (such as STDIN) rather than a file (see [Streaming Data Via JDBC](#)).
2. Use the COPY statement with the FROM LOCAL option.

A non-superuser can also perform a [batch load with a JDBC prepared statement](#), which invokes COPY to load data as a background task.

Extracting Data From an Existing Database

If possible, export the data in text form to a local file or attached disk. When working with large amounts of load data (> 500GB), Vertica recommends that you test the load process using smaller load files as described in [Configuration Procedure](#) to avoid compatibility or file formatting issues.

ETL products typically use ODBC or JDBC to extract data, which gives them program-level access to modify load file column values, as needed.

Database systems typically provide a variety of export methods.

Tip: To export data from an Oracle database, run a SELECT query in Oracle's SQL*Plus command line query tool using a specified column delimiter, suppressed headers, and so forth. Redirect the output to a local file.

Smaller tables generally fit into a single load file. Split any large tables into 250-500GB load files. For example, a 10 TB fact table will require 20-40 load files to maintain performance.

Checking for Delimiter Characters in Load Data

The default delimiter for the COPY statement is a vertical bar (|). Before loading your data, make sure that no CHAR(N) or VARCHAR(N) data values include the delimiter character.

To test for the existence of a specific character in a column, use a query such as this:

```
SELECT COUNT(*) FROM T WHERE X LIKE '%|%'
```

If only a few rows contain |, you can eliminate them from the load file using a WHERE clause and load them separately using a different delimiter.

Tip: For loading data from an Oracle database, use a WHERE clause to avoid problem rows in the main load file, and the negated WHERE clause with REGEX_REPLACE for problem rows.

Moving Data From an Existing Database to Vertica Nodes

To move data from an existing database to Vertica, consider using:

- USB 2.0 (or possibly SATA) disks
- A fast local network connection

Deliver chunks of data to the different Vertica nodes by connecting the transport disk or by writing files from network copy.

Loading From a Local Hard Disk

USB 2.0 disks can deliver data at about 30 MB per second, or 108 GB per hour. USB 2.0 disks are easy to use for transporting data from Linux to Linux. Set up an ext4 filesystem on the disk and write large files there. Linux 2.6 has USB plug-and-play support, so a USB 2.0 disk is instantly usable on various Linux systems.

For other UNIX variants, if there is no common filesystem format available, use the disk without a filesystem to copy a single large file. For example:

```
$ cp bigfile /dev/sdc1
```

Even without a filesystem on the disk, plug-and-play support still works on Linux to provide a device node for the disk. To find out the assigned device, plug in the disk and enter:

```
$ dmesg | tail -40
```

SATA disks are usually internal, but can be external, or unmounted safely if they are internal.

Loading Over the Network

A 1Gbps (gigabits per second) network can deliver about 50 MB/s, or 180GB/hr. Vertica can load about 30-50GB/hour/node for a 1-Ksafe projection design. Therefore, you should use a dedicated 1Gbps LAN. Using a LAN with a performance that is < 1Gbps will be proportionally slower. Vertica recommends not loading data across an external network, because the delays over distance slow down the TCP protocol to a small fraction of its available bandwidth, even without competing traffic.

Note: The actual load rates you obtain can be higher or lower depending on the properties of the data, number of columns, number of projections, and hardware and network speeds. Load speeds can be further improved by using multiple parallel streams.

Loading From Windows

Use NTFS for loading files directly from Windows to Linux. Although Red Hat Linux as originally installed can read Windows FAT32 file systems, this is not recommended.

Using Load Scripts

You can write and run a load script for the `COPY` statement using a simple text-delimited file format. For information about other load formats see [Specifying a COPY Parser](#). Vertica recommends that you load the smaller tables before the largest tables. To check data formats before loading, see [Checking Data Format Before or After Loading](#).

Using Absolute Paths in a Load Script

Unless you are using the `COPY FROM LOCAL` statement, using `COPY` on a remote client requires an absolute path for a data file. You cannot use relative paths on a remote client. For a load script, you can use vsql variables to specify the locations of data files relative to your Linux working directory.

To use vsql variables to specify data file locations:

1. Create a vsql variable containing your Linux current directory.

```
\set t_pwd `pwd`
```

2. Create another vsql variable that uses a path relative to the Linux current directory variable for a specific data file.

```
\set input_file `\:t_pwd'/Date_Dimension.tbl`
```

3. Use the second variable in the `COPY` statement:

```
=> COPY Date_Dimension FROM :input_file DELIMITER '|';
```

4. Repeat steps 2 and 3 to load all data files.

Note: COPY FROM LOCAL does not require an absolute path for data files. You can use paths that are relative to the client's directory system.

Running a Load Script

You can run a load script on any host, as long as the data files are on that host.

1. Change your Linux working directory to the location of the data files.

```
$ cd /opt/vertica/doc/retail_example_database
```

2. Run the Administration Tools.

```
$ /opt/vertica/bin/admintools
```

3. Connect to the database.
4. Run the load script.

Loading Data Interactively

Micro Focus recommends using the COPY statement in script files, as described in [Using Load Scripts](#). You can also load data interactively by piping a text file to vsq1 and executing a COPY (or COPY FROM LOCAL) statement with the standard input stream as the input file.

For example:

```
$ cat fact_table.tbl | vsq1 -c "COPY FACT_TABLE FROM STDIN DELIMITER '|' DIRECT";  
$ cat fact_table.tbl | vsq1 -c "COPY FACT_TABLE FROM LOCAL STDIN DELIMITER '|' DIRECT";
```

Using COPY and COPY LOCAL

The COPY statement bulk loads data into a Vertica database. You can initiate loading one or more files or pipes on a cluster host. You can also load directly from a client system, using the COPY statement with its FROM LOCAL option.

COPY lets you load *parsed* or *computed* data. Parsed data is from a table or schema using one or more columns, and computed data is calculated with a column expression on one or more column values.

COPY invokes different parsers depending on the data format you specify:

- Delimited text (the default parser format, but never specified)
- Native binary (NATIVE) (not supported with COPY LOCAL)
- Native varchar (NATIVE VARCHAR) (not supported with COPY LOCAL)
- Fixed-width data (FIXEDWIDTH)
- ORC (Optimized Row Columnar) and Parquet Hadoop files (not supported with COPY LOCAL)

To use one of the several flex table parsers, use the parser parameter, followed by the parser of choice.

COPY has many options, which you can combine to make importing data flexible. For detailed syntax of the various options see the SQL Reference Manual. For example:

For this option...	See this section...
Read uncompressed data, or data in GZIP, BZIP, or LZO compressed forms.	Specifying COPY FROM Options
Insert data into the WOS (memory) or directly into the ROS (disk).	Choosing a Load Method
Set parameters such as data delimiters and quote characters for the entire load operation or, for specific columns.	Loading UTF-8 Format Data
Transform data before inserting it into the database.	Transforming Data During Loads

Copying Data from a Vertica Client

Use `COPY LOCAL` to load files on a client system to the Vertica database. For example, to copy a GZIP file from your local client, use a command such as this:

```
=> COPY store.store_dimension FROM LOCAL '/usr/files/my_data/input_file' GZIP;
```

You can use a comma-separated list to load multiple files of the same compression type. `COPY LOCAL` then concatenates the files into a single file, so you cannot combine files with different compression types in the list. When listing multiple files, be sure to specify the type of every input file, such as BZIP, as shown:

```
=> COPY simple_table FROM LOCAL 'input_file.bz' BZIP, 'input_file.bz' BZIP;
```

You can load data from a local client from STDIN, as follows:

```
=> COPY simple_table FROM LOCAL STDIN;
```

Loading Data from an IDOL CFS Client

The IDOL Connector Framework Server (CFS) VerticalIndexer feature lets CFS clients connect to your Vertica database using ODBC. After it is connected, CFS uses `COPY . . . FROM LOCAL` statements to load IDOL document metadata into an existing flex table. For more information, see the [Using Flex Tables for IDOL Data](#) section in Using Flex Tables.

Transforming Data During Loads

To promote a consistent database and reduce the need for scripts to transform data at the source, you can transform data with an expression as part of loading. Transforming data while loading is useful for computing values to insert into a target database column from other columns in the source database.

To transform data during load, use the following syntax to specify the target column for which you want to compute values, as an expression:

```
COPY [ [database-name.]schema-name.]table ([[Column as Expression] / column[FORMAT 'format']  
[ ,...]])  
FROM ...
```

Understanding Transformation Requirements

When transforming data during loads, the [COPY](#) statement must contain at least one parsed column. The parsed column can [be a FILLER column](#).

Specify only RAW data in the parsed column source data. If you specify nulls in that RAW data, the columns are evaluated with the same rules as for SQL statement expressions.

You can intersperse parsed and computed columns in a [COPY](#) statement.

Loading FLOAT Values

Vertica parses [floating-point values](#) internally. [COPY](#) does not require you to cast floats explicitly, unless you need to transform the values for another reason.

Using Expressions in COPY Statements

The expression in a [COPY](#) statement can be as simple as a single column, or more complex, such as a case statement for multiple columns. An expression can specify multiple columns, and multiple expressions can refer to the same parsed column. You can use expressions for columns of all supported data types.

[COPY](#) expressions can use many Vertica-supported SQL functions, operators, constants, NULLs, and comments, including these functions:

- [Date/time](#)
-
- [String](#)
- [Null-handling](#)
- [System information](#)

Requirements and restrictions

- [COPY](#) expressions cannot use SQL meta functions ([Vertica-specific](#)), [analytic](#) functions, [aggregate](#) functions, or computed columns.

- For computed columns, you must list all parsed columns in the COPY statement expression. Do not specify FORMAT or RAW in the source data for a computed column.
- Expressions used in a COPY statement can contain only constants. The return data type of the expression must be coercible to that of the target column. Parsed column parameters are also coerced to match the expression.

Handling Expression Errors

Errors in expressions within your COPY statement are SQL errors. As such, they are handled differently from parse errors. When a parse error occurs, COPY rejects the row and adds it to the rejected data file or table. COPY also adds the reason for a rejected row to the exceptions file, or the rejected data table. For example, COPY parsing does not implicitly cast data types. If a type mismatch occurs between the data being loaded and a column type (such as attempting to load a text value into a FLOAT column), COPY rejects the row, and continues processing.

If an error occurs in an expression in your COPY statement, the entire load fails. For example, if your COPY statement has a transform function expression, and a syntax error exists in that expression, the entire load is rolled back. All SQL errors (including COPY rollback from an expression, are stored in the Vertica-specific log file. However, unlike parse rejections and exception messages, SQL expression errors are brief, and may require further research.

Transformation Example

Following is a small transformation example.

1. Create a table t.

```
=> CREATE TABLE t (  
    year VARCHAR(10),  
    month VARCHAR(10),  
    day VARCHAR(10),  
    k timestamp  
);
```

2. Use COPY to copy the table, computing values for the year, month, and day columns in the target database, based on the timestamp columns in the source table.
3. Load the parsed column, timestamp, from the source data to the target database.

```
=> COPY t(year AS TO_CHAR(k, 'YYYY'),
          month AS TO_CHAR(k, 'Month'),
          day AS TO_CHAR(k, 'DD'),
          k FORMAT 'YYYY-MM-DD') FROM STDIN NO COMMIT;
2009-06-17
1979-06-30
2007-11-26
\.
```

4. Select the table contents to see the results:

```
SELECT * FROM t;
 year | month | day | k
-----+-----+-----+-----
 2009 | June  | 17  | 2009-06-17 00:00:00
 1979 | June  | 30  | 1979-06-30 00:00:00
 2007 | November | 26  | 2007-11-26 00:00:00
(3 rows)
```

Deriving Table Columns From Data File Columns

You can use [COPY](#) to derive a table column from the data file to load. For more information, see [Manipulating Source Data Columns](#).

Specifying COPY FROM Options

Each COPY statement requires a FROM option to indicate the location of the file or files being loaded. This syntax excerpt shows the available FROM keywords, and their associated file format options:

```
FROM { STDIN ..... [ BZIP | GZIP | LZO | UNCOMPRESSED ]
... | 'pathToData' [ ON nodename | ON nodeset | ON ANY NODE ]
..... [ BZIP | GZIP | LZO | UNCOMPRESSED ] [, ...]
... | LOCAL STDIN | 'pathToData'
..... [ BZIP | GZIP | LZO | UNCOMPRESSED ] [, ...]
}
```

Each of the FROM keywords lets you optionally specify the format of the load file as UNCOMPRESSED, BZIP, GZIP, or LZO.

Note: When using COPY in conjunction with a CREATE EXTERNAL TABLE statement, you cannot use the COPY FROM STDIN or LOCAL options.

Loading from STDIN

Using STDIN for the FROM option lets you load uncompressed data, BZIP, or GZIP files.

Loading from a Specific Path

You must use the *pathToData* argument to indicate the location of the file to load. You can optionally indicate which node or nodes should parse the input by using any of the following:

- A node name
- A set of nodes
- ON ANY NODE

You can load one or more files, using a path on the local file system or in HDFS.

Using the ON ANY NODE clause indicates that the source file to load is available on all of the nodes. If you specify this clause, COPY opens the file and parses it from any node in the cluster. ON ANY NODE is the default for HDFS paths.

Using the ON *nodeset* clause indicates that the source file is on all named nodes. If you specify this clause, COPY opens the file and parses it from any node in the set. Be sure that the source file you specify is available and accessible on each applicable cluster node.

If *pathToData* resolves to a storage location on a local file system (not HDFS), and the user invoking COPY is not a superuser, these permissions are required:

- The storage location must have been created with the USER option (see [CREATE LOCATION](#))
- The user must already have been granted READ access to the storage location where the file or files exist, as described in [GRANT \(Storage Location\)](#)

If a user with privileges, who is not a superuser, invokes COPY from that storage location, Vertica prevents symbolic links from allowing unauthorized access.

Loading BZIP and GZIP Files

You can load files compressed with BZIP or GZIP. To do so, you must indicate the compression format for each file when loading multiple files. For example, this statement copies a BZIP file into the flex table `twitter`, using the `fjsonparser`:

```
=> COPY twitter FROM '/server1/TWITTER/tweets1.json.bz2' BZIP parser fjsonparser() direct;
Rows Loaded
-----
      172094
(1 row)
```

Loading LZO Files

You can load LZO files using the same COPY statements as you use for BZIP and GZIP files. The following statement loads an LZO file delimited with '|' characters into the flex table twitter:

```
=> COPY twitter FROM '/server1/TWITTER/tweets2.lzo' LZO DELIMITER '|';
Rows Loaded
-----
      19421
(3 rows)
```

Vertica supports the following options to the `lzop` command used to compress the file:

- compression level: -1 through -9, --fast, --best
- -F, --no-checksum, --crc32, --adler32

For more information about these options, see lzop.org.

Loading with Wildcards (glob)

You can invoke COPY for a large number of files in a shared directory with a single statement such as:

```
=> COPY myTable FROM '/data/manyfiles/*.dat' ON ANY NODE;
```

The glob (*) must indicate a set of files, not directories. The following statement fails if `/data/manyfiles` contains any subdirectories:

```
=> COPY myTable FROM '/data/manyfiles/*' ON ANY NODE;
```

Using a wildcard with the `ON ANY NODE` clause expands the file list on the initiator node. This command then distributes the individual files among all nodes, so that the COPY workload is evenly distributed across the entire cluster.

`ON ANY NODE` is the default for HDFS paths, as in the following example:

```
=> COPY myTable FROM 'hdfs:///data/manyfiles/*';
```

You can also distribute a file set across a subset of nodes, which you might do to balance concurrent loads. For example, this command distributes the loading of individual files among the three named nodes:

```
=> COPY myTable FROM '/mydirectory/ofmanyfiles/*.dat'  
    ON (v_vmart_node0001, v_vmart_node0002, v_vmart_node0003);
```

Distributing file loads across nodes depends on two configuration parameters, `EnableApportionLoad` and `EnableApportionFileLoad`. Both are enabled by default. See [General Parameters](#) for more information about these parameters.

Loading from a Local Client

To bulk load data from a client, and without requiring database superuser privileges, use the `COPY FROM LOCAL` option. You can load from either `STDIN`, or a specific path, but not from a specific node (or `ON ANY NODE`), since you are loading from the client. All local files are loaded and parsed serially with each `COPY` statement, so you cannot perform parallel loads with the `LOCAL` option. See [Using Parallel Load Streams](#).

You can load one or more files in the supported formats: `UNCOMPRESSED`, `BZIP`, `GZIP`, or `LZO`.

For specific information about saving rejected data and exceptions files when using `COPY` from `LOCAL`, see [Capturing Load Rejections and Exceptions](#).

Choosing a Load Method

You can specify how Vertica loads table data at two levels:

- Table metadata, through [CREATE TABLE](#) and [ALTER TABLE](#)
- DML statements: [COPY/COPY FROM VERTICA](#), [INSERT](#), [MERGE](#), and [UPDATE](#)

If a table specifies a load option, Vertica uses it for all DML operations unless the DML statement specifies otherwise.

Load Options and Hints

[CREATE TABLE](#) and [ALTER TABLE](#), and copy operations [COPY/COPY FROM VERTICA](#) support the following load options:

- **AUTO** (default): Initially loads data into WOS, suitable for smaller bulk loads.
- **DIRECT**: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads.
- **TRICKLE**: Loads data only into WOS, suitable for frequent incremental loads.

Vertica also supports three load hints: `/*+AUTO*/*`, `/*+DIRECT*/*`, and `/*+TRICKLE*/*`. These hints let you control how individual **INSERT**, **MERGE**, and **UPDATE** operations load table data, overriding the target table's load setting, if any.

AUTO: Loading Into WOS

If no load option is specified for an operation, Vertica uses the **AUTO** method to load data into WOS. After WOS reaches full capacity, Vertica continues to load data to ROS containers.

DIRECT: Loading Directly to ROS

The **DIRECT** option specifies to load data directly into ROS containers, bypassing WOS. **DIRECT** is best suited for large data loads (100 MB or more).

Note: A large initial bulk load can temporarily affect query performance while Vertica organizes the data.

For example:

```
COPY a FROM stdin DIRECT;  
COPY b FROM LOCAL STDIN DIRECT;
```

Tip: Avoid using **DIRECT** to load many smaller data sets. This approach results in many ROS containers that must be combined later.

TRICKLE Loading

Use the **TRICKLE** load option to load data incrementally after the initial bulk load is complete. Trickle loading loads data into the WOS. If the WOS becomes full, an error occurs and the entire data load is rolled back. Use this option only when the following conditions are true:

- You have a finely tuned load and moveout process at your site.
- You are confident that the WOS can hold the data you are loading.

This option is more efficient than AUTO when you want to load data into partitioned tables.

Overriding COPY Auto Commit

By default, COPY automatically commits itself and other current transactions except when loading temporary tables. You can override this behavior by qualifying the COPY statement with the NO COMMIT option. When you specify NO COMMIT, Vertica does not commit the transaction until you explicitly issue a COMMIT statement.

You can use COPY . . . NO COMMIT in two ways:

- Execute multiple COPY commands as a single transaction.
- Check data for constraint violations before committing the load.

Combine Multiple COPY Statements in Same Transaction

When you combine multiple COPY . . . NO COMMIT statements in the same transaction, Vertica can consolidate the data for all operations into fewer ROS containers, and thereby perform more efficiently.

For example, the following set of COPY . . . NO COMMIT statements performs several copy statements sequentially, and then commits them all. In this way, all of the copied data is either committed or rolled back as a single transaction.

```
COPY... NO COMMIT;  
COPY... NO COMMIT;  
COPY... NO COMMIT;  
COPY X FROM LOCAL NO COMMIT;  
COMMIT;
```

Tip: Vertica recommends that you [COMMIT](#) or [ROLLBACK](#) the current transaction before you use COPY . . . NO COMMIT. Otherwise, if a previous operation such as INSERT is in progress, COPY . . . NO COMMIT adds that operation to its own transaction. In this case, the previous operation and copy operation are combined as a single transaction, which requires an explicit COMMIT statement.

Check Constraint Violations

Unless you have enabled enforcement of primary key, unique, or check constraints, Vertica does not check for constraint violations when loading data. You can use `COPY . . . NO COMMIT` to troubleshoot loaded data for constraint violations. Before committing the load, test the data with [ANALYZE_CONSTRAINTS](#). If you find any constraint violations, you can roll back the load.

For details on `ANALYZE_CONSTRAINTS`, see [Detecting Constraint Violations with ANALYZE_CONSTRAINTS](#).

For details on automatic enforcement of primary key, unique, and check constraints, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Specifying a COPY Parser

By default, `COPY` uses the `DELIMITER` parser to load raw data into the database. Raw input data must be in UTF-8, delimited text format. Data is compressed and encoded for efficient storage.

Note: `COPY` cannot explicitly specify the `DELIMITER` parser.

If the raw data to load does not consist primarily of delimited text, specify the parser that is most appropriate, one of the following:

- `NATIVE` (binary)
- `NATIVE VARCHAR`
- `FIXEDWIDTH`
- ORC and PARQUET (see [Reading Hadoop Columnar File Formats](#))

Using a different parser for your data can improve load performance. If delimited input data includes binary data types, `COPY` translates the data on input. See [Using Load Scripts](#) and [Loading Binary \(Native\) Data](#) for examples. You can also load binary data, but only if it adheres to the `COPY` format requirements, described in [Creating Native Binary Format Files](#).

The same bulk load `COPY` statement cannot mix raw data types that require different parsers, such as `NATIVE` and `FIXEDWIDTH`. For information about verifying input data formats, see [Checking Data Format Before or After Loading](#).

Flex Table Parsers

You can use flex parsers to load data into standard, columnar tables. Use the flex parser that best matches your needs, as described in the [Using Flex Table Parsers](#) section of Using Flex Tables:

- FAVROPARSER
- FCEFPARSER
- FCSVPARSER
- FDELIMITEDPARSER
- FDELIMITEDPAIRPARSER
- FJSONPARSER
- FREGEXPARSER

Loading data with the flex parsers makes loading data flexible. For example, you can load JSON data into a columnar table in one load with the FJSONPARSER, and delimited data into the same table in another with the default COPY parser. Using Flex Tables describes this use case and presents an example.

Specifying Load Metadata

In addition to choosing a parser option, COPY supports other options to determine how to handle raw data. These options are considered load metadata, and you can specify metadata options in different parts of the COPY statement as follows:

	Qualifies:		
Metadata option	Column expression	COLUMN OPTION	FROM options
DELIMITER	Y	Y	Y
ENCLOSED BY	Y	Y	Y
ESCAPE AS	Y	Y	Y

	Qualifies:		
Metadata option	Column expression	COLUMN OPTION	FROM options
NULL	Y	Y	Y
TRIM	Y		Y
RECORD TERMINATOR			Y
SKIP			Y
SKIP BYTES			Y (fixed-width only)
TRAILING NULLCOLS			Y

The following precedence rules apply to all data loads:

- All column-level parameters override statement-level parameters.
- COPY uses the statement-level parameter if you do not specify a column-level parameter.
- COPY uses the default metadata values for the DELIMITER, ENCLOSED BY, ESCAPE AS, and NULL options if you do not specify them at either the statement or column level.

When you specify any metadata options, COPY uses the parser to produce the best results and stores the raw data and its corresponding metadata in the following formats:

Raw data format	Metadata format	Parser
UTF-8	UTF-8	DELIMITER
Binary	Binary	NATIVE
UTF-8	Binary	NATIVE VARCHAR
UTF-8	UTF-8	FIXEDWIDTH

Interpreting Last Column End of Row Values

When bulk-loading delimited text data using the default parser (DELIMITED), the last column end of row value can be any of the following:

- Record terminator
- EOF designator
- Delimiter and a record terminator

Note: The FIXEDWIDTH parser always requires exactly a record terminator. No other permutations work.

For example, given a three-column table, the following input rows for a COPY statement using a comma (,) delimiter are each valid:

```
1,1,11
1,1,
1,1,
```

The following examples illustrate how COPY can interpret different last column end of data row values.

Using a Single End of Row Definition

To see how COPY interprets a single end of row definition:

1. Create a two-column table `two_col`, specifying column `b` with a default value of 5:

```
=> CREATE TABLE two_col (a int, b int DEFAULT 5);
CREATE TABLE
```

2. COPY the `two_col` table using a comma (,) delimiter, and enter values for only one column (as a single, multi-line entry):

```
=> COPY two_col from stdin delimiter ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1,
>> 1,
>> \.
```

The COPY statement complete successfully.

3. Query table `two_col`, to display the two NULL values for column `b` as blank:

```
=> SELECT * FROM two_col;
a | b
```

```
----+----  
1 |  
1 |  
(2 rows)
```

Here, COPY expects two values for each column, but gets only one. Each input value is followed by a delimiter (,), and an implicit record terminator (a newline character, \n). You supply a record terminator with the ENTER or RETURN key. This character is not represented on the screen.

In this case, the delimiter (,) and record terminator (\n) are handled independently. COPY interprets the delimiter (,) to indicate the end of one value, and the record terminator (\n) to specify the end of the column row. Since no value follows the delimiter, COPY supplies an empty string before the record terminator. By default, the empty string signifies a NULL, which is a valid column value.

Using a Delimiter and Record Terminator End of Row Definition

To use a delimiter and record terminator together as an end of row definition:

1. Copy column a (a) of the two_col table, using a comma delimiter again, and enter two values:

```
=> COPY two_col (a) FROM STDIN delimiter ',';  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 2,  
>> 2,  
>> \.
```

The COPY statement again completes successfully.

2. Query table two_col to see that column b now includes two rows with its default value (5):

```
=> SELECT * FROM two_col;  
a | b  
----+----  
1 |  
1 |  
2 | 5  
2 | 5  
(4 rows)
```

In this example, COPY expects values for only one column, because of the column (a) directive. As such, COPY interprets the delimiter and record terminator together as a single,

valid, last column end of row definition. Before parsing incoming data, COPY populates column b with its default value, because the table definition has two columns and the COPY statement supplies only one. This example populates the second column with its default column list value, while the previous example used the supplied input data.

Loading UTF-8 Format Data

You can specify these parameters at either a statement or column basis:

- ENCLOSED BY
- ESCAPE AS
- NULL
- DELIMITER

Loading Special Characters As Literals

The default COPY statement escape key is a backslash (\). By preceding any *special character* with an escape character, COPY interprets the character that follows literally, and copies it into the database. These are the special characters that you escape to load them as literals:

Special Character	COPY Statement Usage
Vertical bar ()	Default COPY ... DELIMITER character
Empty string (' ')	Default COPY ... NULL string.
Backslash (\)	Default COPY ... ESC character.
Newline and other control characters	Various

To use a special character as a literal, prefix it with an escape character. For example, to include a literal backslash (\) in the loaded data (such as when including a file path), use two backslashes (\\). COPY removes the escape character from the input when it loads escaped characters.

Using a Custom Column Separator (DELIMITER)

The default COPY delimiter is a vertical bar (|). The DELIMITER is a single ASCII character used to separate columns within each record of a file. Between two delimiters, COPY interprets all

string data in load files as characters. Do not enclose character strings in quotes, since quote characters are also treated as literals between delimiters.

You can define a different delimiter using any ASCII value in the range E'\000' to E'\177' inclusive. For instance, if you are loading CSV data files, and the files use a comma (,) character as a delimiter, you can change the default delimiter to a comma. You cannot use the same character for both the DELIMITER and NULL options.

If the delimiter character is among a string of data values, use the ESCAPE AS character (\ by default) to indicate that the delimiter should be treated as a literal.

The COPY statement accepts empty values (two consecutive delimiters) as valid input data for CHAR and VARCHAR data types. COPY stores empty columns as an empty string (' '). An empty string is not equivalent to a NULL string.

To indicate a non-printing delimiter character (such as a tab), specify the character in extended string syntax (E'...'). If your database has StandardConformingStrings enabled, use a Unicode string literal (U&'...'). For example, use either E'\t' or U&'\0009' to specify tab as the delimiter.

Using a Custom Column Option DELIMITER

This example, redefines the default delimiter through the COLUMN OPTION parameter.

1. Create a simple table.

```
=> CREATE TABLE t(      pk INT,
      co11 VARCHAR(10),
      co12 VARCHAR(10),
      co13 VARCHAR(10),
      co14 TIMESTAMP);
```

2. Use the COLUMN OPTION parameter to change the co11 default delimiter to a tilde (~).

```
=> COPY t COLUMN OPTION(co11 DELIMITER '~') FROM STDIN NO COMMIT;
>> 1|ee~gg|yy|1999-12-12
>> \.
=> SELECT * FROM t;
 pk | co11 | co12 | co13 |          co14
-----+-----+-----+-----+-----
  1 | ee   | gg   | yy   | 1999-12-12 00:00:00
(1 row)
```

Defining a Null Value (NULL)

The default NULL value for COPY is an empty string (' '). You can specify a NULL as any ASCII value in the range E '\001' to E '\177' inclusive (any ASCII character except NUL: E '\000'). You cannot use the same character for both the DELIMITER and NULL options.

When NULL is an empty string (' '), use quotes to insert an empty string instead of a NULL. For example, using NULL " ENCLOSED BY ' " ' :

- 1 | | 3 — Inserts a NULL in the second column.
- 1 | " " | 3 — Inserts an empty string instead of a NULL in the second columns.

To input an empty or literal string, use quotes (ENCLOSED BY); for example:

```
NULL 'NULL 'literal'
```

A NULL is case-insensitive and must be the only value between the data field delimiters. For example, if the null string is NULL and the delimiter is the default vertical bar (|):

| NULL | indicates a null value.

| NULL | does not indicate a null value.

When you use the COPY command in a script, you must substitute a double-backslash for each null string that includes a backslash. For example, the scripts used to load the example database contain:

```
COPY ... NULL E'\\n' ...
```

Loading NULL Values

You can specify NULL by entering fields without content into a data file, using a field delimiter.

For example, given the default delimiter (|) and default NULL (empty string) definition, COPY inserts the following input data:

```
| | 1| 2 | 3  
4 | | 5  
6 | |
```

into the table as follows:

```
(null, null, 1)(null, 2, 3)  
(4, null, 5)
```

```
(6, null, null)
```

If NULL is set as a literal (' null '), COPY inserts the following inputs:

```
null | null | 1null | 2 | 3
4 | null | 5
6 | null | null
```

as follows:

```
(null, null, 1)(null, 2, 3)
(4, null, 5)
(6, null, null)
```

Filling Columns with Trailing Nulls (TRAILING NULLCOLS)

Loading data using the TRAILING NULLCOLS option inserts NULL values into any columns without data. Before inserting TRAILING NULLCOLS, Vertica verifies that the column does not have a NOT NULL constraint.

To use the TRAILING NULLCOLS parameter to handle inserts with fewer values than data columns:

1. Create a table:

```
=> CREATE TABLE z (a INT,
  b INT,
  c INT );
```

2. Insert some values into the table:

```
=> INSERT INTO z VALUES (1, 2, 3);
```

3. Query table z to see the inputs:

```
=> SELECT * FROM z;
 a | b | c
---+---+---
 1 | 2 | 3
(1 row)
```

4. Insert two rows of data from STDIN, using TRAILING NULLCOLS:

```
=> COPY z FROM STDIN TRAILING NULLCOLS;  
>> 4 | 5 | 6  
>> 7 | 8  
>> \.
```

5. Query table z again to see the results. Using TRAILING NULLCOLS, the COPY statement correctly handled the third row of column c, which had no value:

```
=> SELECT * FROM z;  
 a | b | c  
---+---+---  
 1 | 2 | 3  
 4 | 5 | 6  
 7 | 8 |  
(3 rows)
```

Attempting to Fill a NOT NULL Column with TRAILING NULLCOLS

You cannot use TRAILING NULLCOLS on a column that has a NOT NULL constraint. For instance:

1. Create a table n, declaring column b with a NOT NULL constraint:

```
=> CREATE TABLE n ( a INT,  
                    b INT NOT NULL,  
                    c INT );
```

2. Insert some table values:

```
=> INSERT INTO n VALUES (1, 2, 3);  
=> SELECT * FROM n;  
 a | b | c  
---+---+---  
 1 | 2 | 3  
(1 row)
```

3. Use COPY with TRAILING NULLCOLS on table n to see the COPY error due to the column constraint:

```
=> COPY n FROM STDIN trailing nullcols abort on error;  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 4 | 5 | 6  
>> 7 | 8
```

```
>> 9
>> \.
ERROR: COPY: Input record 3 has been rejected (Cannot set trailing column to NULL as column 2
(b) is NOT NULL)
```

4. Query the table to see that the COPY statement values were rejected:

```
=> SELECT * FROM n;
 a | b | c
-----+-----
 1 | 2 | 3
(1 row)
```

Changing the Default Escape Character (ESCAPE AS)

The default escape character is a backslash (\). To change the default to a different character, use the `ESCAPE AS` option. To use an alternative escape character:

```
=> COPY mytable FROM '/data/input.txt' ESCAPE AS E('\001');
```

You can set the escape character to be any ASCII value value in the range `E '\001'` to `E '\177'` inclusive.

Eliminating Escape Character Handling

If you do not want any escape character and want to prevent any characters from being interpreted as escape sequences, use the `NO ESCAPE` option as part of the `COPY` statement.

Delimiting Characters (ENCLOSED BY)

The `COPY ENCLOSED BY` parameter lets you set an ASCII character to delimit characters to embed in string values. You can use any ASCII value in the range `E '\001'` to `E '\177'` inclusive (any ASCII character except `NULL: E '\000'`) for the `ENCLOSED BY` value. Using double quotation marks (") is the most commonly used quotation character. For instance, the following parameter specifies that input data to the `COPY` statement is enclosed within double quotes:

```
ENCLOSED BY '"'
```

With the following input (using the default `DELIMITER (|)` character), specifying:

```
"vertica | value"
```

Results in:

- Column 1 containing "vertica
- Column 2 containing value"

Notice the double quotes (") before `vertica` and after `value`.

Using the following sample input data as follows, columns are distributed as shown:

```
"1", "vertica,value", ",", ""
```

```
col1 | col2          | col3 | col4-----+-----+-----+-----  
1    | vertica,value | ,    | '  '  
(1 row)
```

Alternatively, write the above example using any ASCII character of your choosing:

```
~1~, ~vertica,value~, ~,~, ~'~
```

If you use a single quote ('), rather than double quotes (") as the ENCLOSED BY character, you must escape it using extended string syntax, a Unicode literal string (if `StandardConformingStrings` is enabled), or by using four single quotes:

```
ENCLOSED BY E'\''ENCLOSED BY U&'\0027'  
ENCLOSED BY ''''
```

Using any of the definitions means the following input is properly parsed:

```
'1', 'vertica,value', ',', '\'
```

See [String Literals \(Character\)](#) for an explanation of the string literal formats you can use to specify the ENCLOSED BY parameter.

Use the ESCAPE AS character to embed the ENCLOSED BY delimiter within character string values. For example, using the default ESCAPE AS character (\) and double quote as the ENCLOSED BY character, the following input returns "vertica":

```
"\"vertica\""
```

Using ENCLOSED BY for a Single Column

The following example uses double quotes to enclose a single column (rather than the entire row). The COPY statement also specifies a comma (,) as the delimiter.

```
=> COPY Retail.Dim (Dno, Dname ENCLOSED BY '"', Dstore) FROM '/home/dbadmin/dim3.txt'  
DELIMITER ','  
EXCEPTIONS '/home/dbadmin/exp.txt';
```

This example correctly loads data such as:

```
123,"Smith, John",9832
```

Specifying a Custom End of Record String (RECORD TERMINATOR)

To specify the literal character string that indicates the end of a data file record, use the `RECORD TERMINATOR` parameter, followed by the string to use. If you do not specify a value, then Vertica attempts to determine the correct line ending, accepting either just a linefeed (`E'\n'`) common on UNIX systems, or a carriage return and linefeed (`E'\r\n'`) common on Windows platforms.

For example, if your file contains comma-separated values terminated by line feeds that you want to maintain, use the `RECORD TERMINATOR` option to specify an alternative value:

```
=> COPY mytable FROM STDIN DELIMITER ',' RECORD TERMINATOR E'\n';
```

To specify the `RECORD TERMINATOR` as non-printing characters, use either the extended string syntax or Unicode string literals. The following table lists some common record terminator characters. See [String Literals](#) for an explanation of the literal string formats.

Extended String Syntax	Unicode Literal String	Description	ASCII Decimal
<code>E'\b'</code>	<code>U&'\0008'</code>	Backspace	8
<code>E'\t'</code>	<code>U&'\0009'</code>	Horizontal tab	9
<code>E'\n'</code>	<code>U&'\000a'</code>	Linefeed	10
<code>E'\f'</code>	<code>U&'\000c'</code>	Formfeed	12
<code>E'\r'</code>	<code>U&'\000d'</code>	Carriage return	13
<code>E'\''</code>	<code>U&'\005c'</code>	Backslash	92

If you use the `RECORD TERMINATOR` option to specify a custom value, be sure the input file matches the value. Otherwise, you may get inconsistent data loads.

Note: The record terminator cannot be the same as `DELIMITER`, `NULL`, `ESCAPE`, or `ENCLOSED BY`.

If using JDBC, Micro Focus recommends that you use the following value for the RECORD TERMINATOR:

```
System.getProperty("line.separator")
```

Examples

The following examples use a comma (,) as the DELIMITER for readability.

```
,1,2,3,,1,2,3  
1,2,3,
```

Leading (, 1) and trailing (3,) delimiters are ignored. Thus, the rows all have three columns.

123, '\\n', \\n, 456	Using a non-default null string, the row is interpreted as: 123 newLine \\n 456
----------------------	---

123, this\\, that\\, or the other, something else, 456	The row would be interpreted as: 123 this, that, or the other something else 456
--	--

Loading Binary (Native) Data

You can load binary data using the NATIVE parser option, except with COPY LOCAL, which does not support this option. Since binary-format data does not require the use and processing of delimiters, it precludes the need to convert integers, dates, and timestamps from text to their native storage format, and improves load performance over delimited data. All binary-format files must adhere to the formatting specifications described in [Appendix: Binary File Formats](#).

Native binary format data files are typically larger than their delimited text format counterparts, so compress the data before loading it. The NATIVE parser does not support concatenated compressed binary files. You can load native (binary) format files when developing plug-ins to ETL applications.

There is no copy format to load binary data byte-for-byte because the column and record separators in the data would have to be escaped. Binary data type values are padded and translated on input, and also in the functions, operators, and casts supported.

Loading Hexadecimal, Octal, and Bitstring Data

You can use the formats hexadecimal, octal, and bitstring only to load binary columns. To specify these column formats, use the [COPY](#) statement's `FORMAT` options:

- Hexadecimal
- Octal
- Bitstring

The following examples illustrate how to use the `FORMAT` option.

1. Create a table:

```
=> CREATE TABLE t(oct VARBINARY(5),  
  hex VARBINARY(5),  
  bitstring VARBINARY(5) );
```

2. Create the projection:

```
=> CREATE PROJECTION t_p(oct, hex, bitstring) AS SELECT * FROM t;
```

3. Use a `COPY` statement with the `STDIN` clause, specifying each of the formats:

```
=> COPY t (oct FORMAT 'octal', hex FORMAT 'hex',  
  bitstring FORMAT 'bitstring')  
  FROM STDIN DELIMITER ',';
```

4. Enter the data to load, ending the statement with a backslash (\) and a period (.) on a separate line:

```
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 141142143144145,0x6162636465,0110000101100010011000110110010001100101  
>> \.
```

5. Use a select query on table `t` to view the input values results:

```
=> SELECT * FROM t;  
  oct | hex | bitstring  
-----+-----+-----  
abcde | abcde | abcde  
(1 row)
```

COPY uses the same default format to load binary data, as used to input binary data. Since the backslash character ('\ ') is the default escape character, you must escape octal input values. For example, enter the byte '\141' as '\\141'.

Note: If you enter an escape character followed by an invalid octal digit or an escape character being escaped, COPY returns an error.

On input, COPY translates string data as follows:

- Uses the [HEX_TO_BINARY](#) function to translate from hexadecimal representation to binary.
- Uses the [BITSTRING_TO_BINARY](#) function to translate from bitstring representation to binary.

Both functions take a VARCHAR argument and return a VARBINARY value.

You can also use the escape character to represent the (decimal) byte 92 by escaping it twice; for example, '\\\\'. Note that vsql inputs the escaped backslash as four backslashes. Equivalent inputs are hex value '\0x5c' and octal value '\134' ($134 = 1 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 92$).

You can load a delimiter value if you escape it with a backslash. For example, given delimiter '|', '\\\001\\|\002' is loaded as {1,124,2}, which can also be represented in octal format as '\\\001\\174\\002'.

If you insert a value with more bytes than fit into the target column, COPY returns an error. For example, if column c1 is VARBINARY(1):

```
=> INSERT INTO t (c1) values ('ab'); ERROR: 2-byte value too long for type Varbinary(1)
```

If you implicitly or explicitly cast a value with more bytes than fit the target data type, COPY silently truncates the data. For example:

```
=> SELECT 'abcd'::binary(2);
binary
-----
ab
(1 row)
```

Hexadecimal Data

The optional '\0x' prefix indicates that a value is hexadecimal, not decimal, although not all hexadecimal values use A-F; for example, 5396. COPY ignores the \0x prefix when loading the input data.

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low nibble of the first (furthest to the left) byte.

Octal Data

Loading octal format data requires that each byte be represented by a three-digit octal code. The first digit must be in the range [0,3] and the second and third digits must both be in the range [0,7].

If the length of an octal value is not a multiple of three, or if one of the three digits is not in the proper range, the value is invalid and COPY rejects the row in which the value appears. If you supply an invalid octal value, COPY returns an error. For example:

```
SELECT '\\000\\387'::binary(8);  
ERROR: invalid input syntax for type binary
```

Rows that contain binary values with invalid octal representations are also rejected. For example, COPY rejects '\\008' because '\\ 008' is not a valid octal number.

BitString Data

Loading bitstring data requires that each character must be zero (0) or one (1), in multiples of eight characters. If the bitstring value is not a multiple of eight characters, COPY treats the first n characters as the low bits of the first byte (furthest to the left), where n is the remainder of the value's length, divided by eight.

Examples

The following example shows VARBINARY HEX_TO_BINARY(VARCHAR) and VARCHAR TO_HEX(VARBINARY) usage.

1. Create table `t` and its projection with binary columns:

```
=> CREATE TABLE t (c BINARY(1));  
=> CREATE PROJECTION t_p (c) AS SELECT c FROM t;
```

2. Insert minimum and maximum byte values, including an IP address represented as a character string:

```
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
```

```
=> INSERT INTO t values (V6_ATON('2001:DB8::8:800:200C:417A'));
```

Use the `TO_HEX` function to format binary values in hexadecimal on output:

```
=> SELECT TO_HEX(c) FROM t;
to_hex
-----
00
ff
20
(3 rows)
```

See Also

- [COPY](#)
- [Binary Data Types](#)
-
- [ASCII](#)

Loading Native Varchar Data

Use the `NATIVE VARCHAR` parser option when the raw data consists primarily of `CHAR` or `VARCHAR` data. `COPY` performs the conversion to the actual table data types on the database server. This parser option is not supported with `COPY LOCAL`.

Using `NATIVE VARCHAR` does not provide the same efficiency as `NATIVE`. However, `NATIVE VARCHAR` precludes the need to use delimiters or to escape special characters, such as quotes, which can make working with client applications easier.

Note: `NATIVE VARCHAR` does not support concatenated compressed files.

Batch data inserts performed through the Vertica ODBC and JDBC drivers automatically use the `NATIVE VARCHAR` format.

Loading Fixed-Width Format Data

Use the `FIXEDWIDTH` parser option to bulk load fixed-width data. You must specify the `COLSIZES` option values to specify the number of bytes for each column. The definition of the

table you are loading (`COPY table f (x, y, z)`) determines the number of `COLSIZES` values to declare.

To load fixed-width data, use the `COLSIZES` option to specify the number of bytes for each input column. If any records do not have values, `COPY` inserts one or more null characters to equal the specified number of bytes. The last record in a fixed-width data file must include a record terminator to determine the end of the load data.

Supported Options for Fixed-Width Data Loads

Loading fixed-width data supports the options listed in the [COPY Option Parser Dependencies](#).

These options are not supported:

- `DELIMITER`
- `ENCLOSED BY`
- `ESCAPE AS`
- `TRAILING NULLCOLS`

Using Nulls in Fixed-Width Data

The default `NULL` string for a fixed-width load cannot be an empty string, and instead, consists of all spaces. The number of spaces depends on the column width declared with the `COLSIZES (integer, [, ...])` option.

For fixed-width loads, the `NULL` definition depends on whether you specify `NULL` at the column or statement level:

- *Statement level:* `NULL` must be defined as a single-character. The default (or custom) `NULL` character is repeated for the entire width of the column.
- *Column level:* `NULL` must be defined as a string whose length matches the column width.

For fixed-width loads, if the input data column has fewer values than the specified column size, `COPY` inserts `NULL` characters. The number of `NULL`s must match the declared column width. If you specify a `NULL` string at the column level, `COPY` matches the string with the column width.

Note: To turn off `NULL`s, use the `NULL AS` option and specify `NULL AS ''`.

Defining a Null Character (Statement Level)

1. Create a two-column table (fw):

```
=> CREATE TABLE fw(co int, ci int);  
CREATE TABLE
```

2. Copy the table, specifying null as 'N', and enter some data:

```
=> COPY fw FROM STDIN FIXEDWIDTH colsizes(2,2) null AS 'N' NO COMMIT;  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> NN12  
>> 23NN  
>> NNNN  
>> nnnn  
>> \.
```

3. Select all (*) from the table:

```
=> SELECT * FROM fw;  
co | ci  
----+----  
23 |  
   |  
   | 12  
(4 rows)
```

Defining a Custom Record Terminator

To define a record terminator other than the COPY default when loading fixed-width data, take these steps:

1. Create table fw with two columns, co and ci:

```
=> CREATE TABLE fw(co int, ci int);  
CREATE TABLE
```

2. Copy table fw, specifying two 2-byte column sizes, and specifying a comma (,) as the record terminator:

```
=> COPY fw FROM STDIN FIXEDWIDTH colsizes(2,2) RECORD TERMINATOR ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1234,1444,6666
>> \.
```

3. Query all data in table fw:

```
=> SELECT * FROM fw;
 co | ci
-----+-----
 12 | 34
 14 | 44
(2 rows)
```

The SELECT output indicates only two values. COPY rejected the third value (6666) because it was not followed by a comma (,) record terminator. Fixed-width data requires a trailing record terminator only if you explicitly specify a record terminator explicitly.

Copying Fixed-Width Data

Use COPY FIXEDWIDTH COLSIZES (n [, ...]) to load files into a Vertica database. By default, all spaces are NULLs. For example:

```
=> CREATE TABLE mytest(co int, ci int);
=> CREATE PROJECTION mytest_p1 AS SELECT * FROM mytest SEGMENTED BY HASH(co) ALL NODES;
=> COPY mytest(co,ci) FROM STDIN FIXEDWIDTH colsizes(6,4) NO COMMIT;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> \.
=> SELECT * FROM mytest ORDER BY co;
 co | ci
-----+-----
(0 rows)
```

Skipping Content in Fixed-Width Data

The COPY statement has two options to skip input data. The SKIP BYTES option is only for fixed-width data loads:

SKIP BYTES <i>num-bytes</i>	Skips the specified number of bytes from the input data.
SKIP <i>num-records</i>	Skips the specified number of records.

The following example uses SKIP BYTES to skip 11 bytes when loading a fixed-width table with two columns (4 and 6 bytes):

1. Copy a table using SKIP BYTES:

```
=> COPY fw FROM STDIN FIXEDWIDTH colsizes (4,6) SKIP BYTES 11;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 2222666666
>> 1111999999
>> 1632641282
>> \.
```

2. Query all data in table fw:

```
=> SELECT * FROM fw ORDER BY co;
  co |  ci
-----+-----
 1111 | 999999
 1632 | 641282
(2 rows)
```

The output confirms that COPY skipped the first 11 bytes of loaded data.

The following example uses SKIP when loading a fixed-width (4,6) table:

1. Copy a table, using SKIP to skip two records of input data:

```
=> COPY fw FROM STDIN FIXEDWIDTH colsizes (4,6) SKIP 2;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 2222666666
>> 1111999999
>> 1632641282
>> 3333888888
>> \.
```

2. Query all data in table fw:

```
=> SELECT * FROM fw ORDER BY co;
  co |  ci
-----+-----
 1632 | 641282
 3333 | 888888
(2 rows)
```

The output confirms that COPY skipped the first two records of load data.

Trimming Characters in Fixed-Width Data Loads

Use the TRIM option to trim a character. TRIM accepts a single-byte character, which is trimmed at the beginning and end of the data. For fixed-width data loads, when you specify a TRIM character, COPY first checks to see if the row is NULL. If the row is not null, COPY trims the character(s). The next example instructs COPY to trim the character A, and shows the results:

1. Copy table fw, specifying TRIM character A:

```
=> COPY fw FROM STDIN FIXEDWIDTH colsizes(4,6) TRIM 'A';  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> A22A444444  
>> A22AA44444A  
>> \.
```

2. Query all data in table fw:

```
=> SELECT * FROM fw ORDER BY co;  
co | ci  
----+-----  
22 | 4444  
22 | 444444  
(2 rows)
```

Using Padding in Fixed-Width Data Loads

By default, the padding character is ' ' (a single space). The padding behavior for fixed-width data loads is similar to how a space is treated in other formats, differing by data type as follows:

Data type	Padding
Integer	Leading and trailing spaces
Bool	Leading and trailing spaces
Float	Leading and trailing spaces
[var]Binary	None, all characters are significant.
[Var]Char	Trailing spaces if string is too large

Data type	Padding
DateInterval Time Timestamp TimestampTZ TimeTZ	None, all characters are significant. COPY uses an internal algorithm to parse these data types.
Date (formatted)	Use the COPY FORMAT option string to match the expected column length.
Numerics	Leading and trailing spaces

Bulk Loading and Exporting Data From Amazon S3

The Vertica library for Amazon Web Services (AWS) is a set of functions and configurable session parameters. These parameters allow you to directly exchange data between Vertica and Amazon S3 storage without any third-party scripts or programs.

To use the AWS library, you must have access to an Amazon S3 storage account.

Related Topics

- [Configure the AWS Library](#)
- [Loading Data From Amazon S3](#)
- [Exporting Data to Amazon S3 From Vertica](#)

Configuring the Vertica Library for Amazon Web Services

Configure the Vertica library for Amazon Web Services (AWS) by setting session parameters with your AWS access key credentials and region. You can set your session parameters directly, or you can store your credentials in a table and set them with the `AWS_SET_CONFIG` function.

Because the AWS library is configured with session parameters, you must reconfigure the library with each new session.

Important: Your AWS access key ID and secret access key are different from your account access credentials. For more information about AWS access keys, visit the [AWS documentation](#).

Use either of the following methods to securely set and store your AWS account credentials:

- [Configuring Session Parameters Directly](#)
- [Configuring Session Parameters Using Keychains](#)

Note: To increase security, configure session parameters directly to avoid storing credentials within Vertica.

AWS Access Key Requirements

In order to communicate with AWS, your access key must have the following permissions:

- s3:GetObject
- s3:PutObject
- s3:ListBucket

For security purposes, Micro Focus International plc recommends that you create a separate access key with limited permissions specifically for use with the Vertica Library for AWS.

Configuring Session Parameters Directly

Set the following session parameters for AWS using your own credentials:

- `aws_id` — This value is your AWS access key ID.

```
=> ALTER SESSION SET UDPARAMETER FOR awslib aws_id='AKABCOEXAMPLEPKPXYZQ';
```

- `aws_secret` — This value is your AWS secret access key.

```
=> ALTER SESSION SET UDPARAMETER FOR awslib aws_secret='CEXAMPLE3tEXAMPLE1wEXAMPLEFrFEXAMPLE6+Yz';
```

- `aws_region` - This value is the AWS region associated with the S3 bucket you intend to access. Left unconfigured, `aws_region` will default to `us-east-1`. It identifies the default

server used by Amazon S3.

```
=> ALTER SESSION SET UDPARAMETER FOR awslib aws_region='us-east-1';
```

Important: Parameter values are case sensitive.

Configuring Session Parameters Using Credentials Stored in a Table

You can place your credentials in a table and secure them with a [row-level access policy](#). You can then call your credentials with the `AWS_SET_CONFIG` scalar meta-function. This approach allows you to store your credentials on your cluster for future session parameter configuration. You must have `dbadmin` access to create access policies.

1. Create a table with rows or columns corresponding with your credentials:

```
=> CREATE TABLE keychain(accesskey varchar, secretaccesskey varchar);
```

2. Store your credentials in the corresponding columns:

```
=> COPY keychain FROM STDIN;  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> AEXAMPLEI5EXAMPLEYXQ|CCEXAMPLEtFjTEXAMPLEiEXAMPLE6+Yz  
>> \.
```

3. Set a [row-level access policy](#) appropriate to your security situation.
4. With each new session, configure your session parameters by calling the `AWS_SET_CONFIG` parameter in a `SELECT` statement:

```
=> SELECT AWS_SET_CONFIG('aws_id', accesskey), AWS_SET_CONFIG('aws_secret', secretaccesskey)  
FROM keychain;  
aws_set_config | aws_set_config  
-----+-----  
aws_id        | aws_secret  
(1 row)
```

Important: Micro Focus International plc recommends that you *not* use the `AWS_SET_CONFIG` function to configure your library directly. When you do so, your credentials are recorded in the server log.

5. After you have configured your session parameters, verify them:

```
=> SHOW SESSION UDPARAMETER ALL;
```

Related Topics

- [Import Data from Amazon S3 Using the Vertica AWS Library](#)
- [Export Data to Amazon S3 Using the Vertica AWS Library](#)
- [AWS_SET_CONFIG](#)
- [AWS_GET_CONFIG](#)

Loading Data From Amazon S3

After you configure the Vertica library for Amazon Web Services (AWS), you can copy data from S3. To do so, use COPY with the S3 UDSOURCE containing the location of your S3 bucket and object. You can use either a standard HTTPS URL, or the S3 URL scheme, as the following examples show.

Use COPY with a standard HTTPS URL:

```
=> COPY exampleTable SOURCE S3(url='https://s3.amazonaws.com/exampleBucket/object');
```

Use COPY with the S3 URL scheme:

```
=> COPY exampleTable SOURCE S3(url='s3://exampleBucket/object');
```

You can use the S3 UDSOURCE with any [UDParser](#) or [UDFilter](#) to import any data format supported by Vertica. If Vertica encounters an error with AWS during an S3 import operation which it cannot resolve, it will abort the import and pass the AWS error information on in the Vertica error report.

Importing Multiple Specific Files

Import multiple specific files by separating the URLs with a bar:

```
=> COPY exampleTable SOURCE s3(url='s3://exampleBucket/object1|S3://exampleBucket/object2');
```

Specify your own delimiter:

```
=> COPY exampleTable SOURCE S3(url='s3://exampleBucket/object1,s3://exampleBucket/object2',  
delimiter=',');
```

Importing Multiple Files Using Glob Expansion

In addition to importing single and multiple files by specifying exact URL addresses and the URL parameter, you can use glob expansion to import multiple files in a bucket or subdirectory by specifying the location with the bucket parameter.

Import from all files in a bucket using glob expansion:

```
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/*');
```

Import from all files in a subdirectory using glob expansion:

```
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/subdirectory/*');
```

Import all files with a 'db_' prefix:

```
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/db_*');
```

Import all files with a .csv suffix:

```
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/*.csv');
```

Related Topics

- [Configuring the Vertica AWS Library](#)
- [Exporting Data to Amazon S3 From Vertica](#)
- [S3 UDSOURCE](#)

Exporting Data to Amazon S3 From Vertica

After you configure the library for Amazon Web Services (AWS), you can export Vertica data to Amazon S3 by calling the S3EXPORT() transform function. S3EXPORT() writes data to files, based on the URL you provide. Vertica performs all communication over HTTPS, regardless of the URL type you use. Vertica does not support virtual host style URLs. If you use HTTPS URL constructions, you must use path style URLs.

Note: If your S3 bucket contains a period in its path, then you should set the `prepend_hash` parameter to `True`.

You can control the output of `S3EXPORT()` in the following ways:

- [Adjust the base query provided to S3EXPORT](#)
- [Adjust the parameters of S3EXPORT](#)
- [Adjust the partition of your result set with the OVER\(\) clause](#)

Adjust the Query Provided to S3EXPORT

By adjusting the query given to `S3EXPORT()`, you can export anything from tables to reporting queries.

This example exports a whole table:

```
=> SELECT S3EXPORT( * USING PARAMETERS url='s3://exampleBucket/object') OVER(PARTITION BEST) FROM
exampleTable;
 rows | url
-----+-----
   606 | s3://exampleBucket/object
(1 row)
```

This example exports the results of a query:

```
=> SELECT S3EXPORT(customer_name, annual_income USING PARAMETERS url='s3://exampleBucket/object')
OVER()
  FROM public.customer_dimension
  WHERE (customer_gender, annual_income) IN
  (SELECT customer_gender, MAX(annual_income)
   FROM public.customer_dimension
   GROUP BY customer_gender);

 rows | url
-----+-----
   606 | s3://exampleBucket/object
(1 row)
```

Adjust the Parameters of S3EXPORT

Vertica exports partition data in chunks set by the `chunksizes` parameter. In its default setting, this multipart parameter instructs AWS to recombine the chunks into a single S3 object. However, if your export exceeds the S3 object size limit, or you want to fix the size of output objects, you can disable multipart uploads. Disabling allows you to set chunking to any size you

prefer under the AWS PUT limit. This approach causes each partition to write your export into multiple incrementing S3 objects.

This example shows how you can export a 27 MB file with multipart disabled and chunk size left to the 10 MB default:

```
=> SELECT S3EXPORT(* USING PARAMETERS url='s3://examplebucket/object',
                        multipart=false) OVER()
        FROM exampleTable;
rows  | url
-----+-----
1048576 | s3://exampleBucket/object
(1 row)
```

Results:

```
10244k  object.699b7f63.1
10244k  object.699b7f63.2
6152k   object.699b7f63.3
```

The base name of the resulting objects matches the provided URL. However, the partition is now split into three parts with a maximum chunk size of 10 MB.

Adjust the Partition of Your Result Set with the OVER Clause

Use the OVER clause to control your export partitions. Using the OVER() clause without qualification results in a single partition processed by the initiator for all of the query data. This example shows how to call the function with an unqualified OVER() clause:

```
=> SELECT S3EXPORT(name, company USING PARAMETERS url='s3://exampleBucket/object',
                        delimiter=',') OVER()
        FROM exampleTable WHERE company='Vertica';
rows  | url
-----+-----
606   | s3://exampleBucket/object
(1 row)
```

You can also use window clauses, such as [window partition clauses](#) and [window order clauses](#), to manage exported objects.

This example shows how you can use a window partition clause to partition S3 objects based on company values:

```
=> SELECT S3EXPORT(name, company
                USING PARAMETERS url='s3://exampleBucket/object',
                delimiter=',') OVER(PARTITION BY company) AS MEDIAN
        FROM exampleTable;
```

Related Topics

- [Configure the AWS Library](#)
- [Loading Data From Amazon S3](#)
- [s3export Function](#)

Manipulating Source Data Columns

When loading data, your source data might contain one or more columns that do not exist in the target table. Or, the source and target tables have matched columns, but you want to omit one or more source columns from the target table.

The [COPY](#) command's `FILLER` parameter helps you accomplish these tasks:

- Ignore input columns from the source data.
- Transform source data before loading it into the target table.

The `FILLER` parameter identifies a column of source data that the `COPY` command can ignore, or use to compute new values that are loaded into the target table. Define the `FILLER` parameter data type so it is compatible with the source data. For example, be sure to define a `VARCHAR` so its length can contain all source data; otherwise, data could be truncated. You can specify multiple filler columns, where each filler column is specified by its own `FILLER` parameter.

Use FILLER to Ignore Some Values and Compute New Values for the Target Table

Using SQL operators or functions, you can combine two or more source columns into one column; you can also split one column into multiple columns—for example, split date strings into day, month, and year components and load these into target table columns.

The following `COPY` statement reads all of the source data, but only loads the source columns `first_name` and `last_name`. It constructs the data for `full_name` by concatenating each of the source data columns. To do this, use the `FILLER` parameter to ignore the `middle_name`

column on load, but use the column when concatenating data to populate the `full_name` column.

```
=> CREATE TABLE names(first_name VARCHAR(20), last_name VARCHAR(20), full_name VARCHAR(60));
CREATE TABLE
=> COPY names(first_name,
              middle_name FILLER VARCHAR(20),
              last_name,
              full_name AS first_name||' '||middle_name||' '||last_name)
FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Marc|Gregory|Smith
>> Sue|Lucia|Temp
>> Jon|Pete|Hamilton
>> \.
=> SELECT * FROM names;
 first_name | last_name |      full_name
-----+-----+-----
 Jon       | Hamilton | Jon Pete Hamilton
 Marc      | Smith    | Marc Gregory Smith
 Sue       | Temp     | Sue Lucia Temp
(3 rows)
```

If the source field's data type is `VARCHAR`, be sure to set the `VARCHAR` length to ensure that the combined length of all `FILLER` source fields does not exceed the target column's defined length; otherwise, the `COPY` command could return with an error.

Using Parallel Load Streams

Vertica can divide the work of loading data, taking advantage of parallelism to speed up the operation. Vertica supports several types of parallelism:

- **Distributed load:** Files in a multi-file load are loaded on several nodes in parallel, instead of all being loaded on a single node.
- **Apportioned load:** A single large file or other single source is divided into segments (portions), which are assigned to several nodes to be loaded in parallel.

Caution: If the file is different on two nodes, an incorrect or incomplete result is returned, with no error or warning.

Apportioned load is enabled by default. If you want to disable it, set the `EnableApportionLoad` configuration parameter to 0.

- **Cooperative parse:** A source being loaded on a single node uses multi-threading to parallelize the parse. Cooperative parse is enabled by default. If you want to disable it, set

the EnableCooperativeParse configuration parameter to 0.

See [General Parameters](#) for information about the configuration parameters.

Specifying Distributed File Loads

The COPY FROM statement provides several ways to distribute a file load.

You can direct individual files in a multi-file load to specific nodes, as in the following example of distributed load.

```
=> COPY t FROM '/data/file1.dat' ON v_vmart_node0001, '/data/file2.dat' ON v_vmart_node0002;
```

You can use globbing (wildcard expansion) to specify a group of files with the ON ANY NODE directive, as in the following example.

- If apportioned load is enabled (the default), Vertica assigns different files to different nodes. Both the EnableApportionedLoad and EnableApportionedFileLoad must be set to 1.
- If apportioned load is disabled, a single node loads all the data.

```
=> COPY t FROM '/data/*.dat' ON ANY NODE;
```

If you have a single file instead of a group of files, you can still, potentially, benefit from apportioned load. The file must be large enough to divide into portions at least equal to ApportionedFileMinimumPortionSizeKB in size. You must also use a parser that supports apportioned load. The delimited parser built into Vertica supports apportioned load, but other parsers might not.

The following example shows how you can load a single large file using multiple nodes.

```
=> COPY t FROM '/data/bigfile.dat' ON ANY NODE;
```

You can limit the nodes that participate in an apportioned load. Doing so is useful if you need to balance several concurrent loads. Vertica apportions each load individually; it does not account for other loads that might be in progress on those nodes. You can, therefore, potentially speed up your loads by managing apportioning yourself.

The following example shows how you can apportion loads on specific nodes.

```
=> COPY t FROM '/data/big1.dat' ON (v_vmart_node0001, v_vmart_node0002, v_vmart_node0003),  
      '/data/big2.dat' ON (v_vmart_node0004, v_vmart_node0005);
```

Loaded files can be of different formats, such as BZIP, GZIP, and others. However, because file compression is a filter, you cannot use apportioned load for a compressed file.

Specifying Distributed Loads with Sources

You can also apportion loads using COPY WITH SOURCE. You can create sources and parsers with the User-Defined Load (UDL) API. If both the source and parser support apportioned load, and EnableApportionLoad is set, then Vertica attempts to divide the load among nodes.

The following example shows a load that you could apportion.

```
=> COPY t WITH SOURCE MySource() PARSER MyParser();
```

The built-in delimited parser supports apportioning, so you can use it with a user-defined source, as in the following example.

```
=> COPY t WITH SOURCE MySource();
```

Number of Load Streams

Although the number of files you can load is not restricted, the optimal number of load streams depends on several factors, including:

- Number of nodes
- Physical and logical schemas
- Host processors
- Memory
- Disk space

Using too many load streams can deplete or reduce system memory required for optimal query processing. See [Best Practices for Managing Workload Resources](#) for advice on configuring load streams.

Monitoring COPY Loads and Metrics

You can check COPY loads using:

- Vertica functions
- LOAD_STREAMS system table
- LOAD_SOURCES system table

Using Vertica Functions

Two meta-functions return COPY metrics for the number of accepted or rejected rows from a COPY statement:

1. To get the number of accepted rows, use the [GET_NUM_ACCEPTED_ROWS](#) function:

```
=> select get_num_accepted_rows();
get_num_accepted_rows
-----
                        11
(1 row)
```

2. To check the number of rejected rows, use the [GET_NUM_REJECTED_ROWS](#) function:

```
=> select get_num_rejected_rows();
get_num_rejected_rows
-----
                        0
(1 row)
```

Note: GET_NUM_ACCEPTED_ROWS and GET_NUM_REJECTED_ROWS support loads from STDIN, COPY LOCAL from a Vertica client, or a single file on the initiator. You cannot use these functions for multi-node loads.

Using the CURRENT_LOAD_SOURCE Function

You can include the CURRENT_LOAD_SOURCE function as a part of the COPY statement. Doing so allows you to insert into a column the input file name or value computed by this function.

To insert the file names into a column from multiple source files:

```
=> COPY t (c1, c2, c3 as CURRENT_LOAD_SOURCE()) FROM '/home/load_file_1' ON exampledb_node02,
'/home/load_file_2' ON exampledb_node03 DELIMITER ',';
```

Using the LOAD_STREAMS System Table

Vertica includes a set of system tables that include monitoring information, as described in [Using System Tables](#). The `LOAD_STREAMS` system table includes information about load stream metrics from `COPY` and `COPY FROM VERTICA` statements. Thus, you can query table values to get `COPY` metrics.

To see all table columns:

```
=> SELECT * FROM load_streams;
```

Using the STREAM NAME Parameter

Using the `STREAM NAME` parameter as part of the `COPY` statement labels `COPY` streams explicitly so they are easier to identify in the `LOAD_STREAMS` system table.

To use the `STREAM NAME` parameter:

```
=> COPY mytable FROM myfile DELIMITER '|' DIRECT STREAM NAME 'My stream name';
```

The `LOAD_STREAMS` system table includes stream names for every `COPY` statement that takes more than 1-second to run. The 1-second duration includes the time to plan and execute the statement.

Vertica maintains system table metrics until they reach a designated size quota (in kilobytes). This quota is set through internal processes, which you cannot set or view directly.

Other LOAD_STREAMS Columns for COPY Metrics

These `LOAD_STREAMS` system table column values depend on the load status:

- `ACCEPTED_ROW_COUNT`
- `REJECTED_ROW_COUNT`
- `PARSE_COMPLETE_PERCENT`
- `SORT_COMPLETE_PERCENT`

When a COPY statement using the DIRECT option is in progress, the ACCEPTED_ROW_COUNT value can increase during parsing. This value can reach the maximum number of rows in the input file.

If COPY reads input data from multiple named pipes, the PARSE_COMPLETE_PERCENT value remains at zero (0) until *all* named pipes return an EOF. While COPY awaits an EOF from multiple pipes, it can appear to be hung. However, before canceling the COPY statement, check your [system CPU and disk accesses](#) to determine if any activity is in progress.

In a typical load, the PARSE_COMPLETE_PERCENT value can either increase slowly or jump quickly to 100%, if you are loading from named pipes or STDIN. However, SORT_COMPLETE_PERCENT remains at 0 when loading from named pipes or STDIN. After PARSE_COMPLETE_PERCENT reaches 100%, SORT_COMPLETE_PERCENT increases to 100%. Depending on the data sizes, a significant lag can occur between the time PARSE_COMPLETE_PERCENT reaches 100% and the time SORT_COMPLETE_PERCENT begins to increase.

This example shows how you can set the VSQL expanded display and then select various columns of data from the LOAD_STREAMS system table:

```
=> \pset expanded
Expanded display is on.
=> SELECT stream_name, table_name, load_start, accepted_row_count,
        rejected_row_count, read_bytes, unsorted_row_count, sorted_row_count,
        sort_complete_percent FROM load_streams;
-[ RECORD 1 ]-----+-----
stream_name      | fact-13
table_name       | fact
load_start       | 2010-12-28 15:07:41.132053
accepted_row_count | 900
rejected_row_count | 100
read_bytes       | 11975
input_file_size_bytes | 0
parse_complete_percent | 0
unsorted_row_count | 3600
sorted_row_count  | 3600
sort_complete_percent | 100
```

See the [SQL Reference Manual](#) for other meta-function details.

Using the LOAD_SOURCES System Table

The LOAD_STREAMS table shows the total number of rows that were loaded or rejected. Grouping this information by source can help you determine from where data is coming. The [LOAD_SOURCES](#) system table includes some of the same data as LOAD_STREAMS does but adds this source-specific information. If apportioning is enabled, [LOAD_SOURCES](#) also provides information about how loads are apportioned among nodes.

You can use this table to identify causes of differing query results. For example, you can use the following statement to create an external table based on globs:

```
=> CREATE EXTERNAL TABLE tt AS COPY WITH SOURCE AWS(dir = 'foo', file = '*');
```

If you select from this table, Vertica loads data from every file in the `foo` directory and creates one row in the `LOAD_SOURCES` table for each file. Suppose you later repeat the query and see different results. You could look at the `LOAD_SOURCES` table and discover that—between the two queries—somebody added another file to the `foo` directory. Because each file is recorded in `LOAD_SOURCES`, you can see the new file that explains the changed query results.

If you are using many data sources, you might prefer to disable this reporting. To disable reporting, set the `LoadSourceStatisticsLimit` configuration parameter to 0. This parameter sets the upper bound on the number of sources profiled by `LOAD_SOURCES` per load. The default value is 256.

Capturing Load Rejections and Exceptions

Loading data with `COPY` has two main phases, *parsing* and *loading*. Rejected data is created whenever `COPY` cannot parse a row of data. Following are some parser errors that can cause a rejected row:

- Unsupported parser options
- Incorrect data types for the table into which data is being loaded
- Malformed context for the parser in use
- Missing delimiters

Other problems can occur during the load phase, but such problems are not rejected data from parser errors.

Several optional parameters let you determine how strictly `COPY` handles rejections when parsing data. For example, you can have `COPY` fail when it rejects a single row, or allow a specific number of parsing rejections before the load fails. This section presents the parameters to determine how `COPY` handles rejected data.

Save Rejected Rows (REJECTED DATA and EXCEPTIONS)

The `COPY` statement automatically saves a copy of each rejected row in a *rejected-data* file. `COPY` also saves a corresponding explanation of what caused the rejection in an *exceptions* file. By default, Vertica saves both files in a database catalog subdirectory, called `CopyErrorLogs`, as listed in this example:

```
v_mart_node003_catalog\COPYErrorLogs\trans-STDIN-copy-from-rejected-data.1  
v_mart_node003_catalog\COPYErrorLogs\trans-STDIN-copy-from-exceptions.1
```

You can optionally save COPY rejections and exceptions in two other ways:

- Use the REJECTED DATA *reject_file* and EXCEPTIONS *except_file* parameters to save both files to a location of your choice. The *reject_file* includes rejected rows, while the exceptions file contains a description of why each row was rejected.
- Use the REJECTED DATA AS TABLE *reject_table* clause. Saving rejected rows to a *reject_table* also retains the exception descriptions.

Note: Vertica recommends saving rejected data to a table. However, saving to a table excludes saving to a default or specific rejected data file.

If you save COPY rejected data to a table, the table files are stored in the data subdirectory. For example, in a VMart database installation, rejected data table records are stored in the RejectionTableData directory as follows:

```
=> cd v_mart_node003_data\RejectionTableData\  
=> ls  
TABLE_REJECTED_RECORDS_"bg"_mytest01.example.-25441:0x6361_45035996273805099_1.1  
TABLE_REJECTED_RECORDS_"bg"_mytest01.example.-25441:0x6361_45035996273805113_2.2  
.  
.  
.  
TABLE_REJECTED_RECORDS_"delimr"_mytest01.example.-5958:0x3d47_45035996273815749_1.1  
TABLE_REJECTED_RECORDS_"delimr"_mytest01.example.-5958:0x3d47_45035996273815749_1.2
```

COPY LOCAL Rejected Data

For COPY LOCAL operations, you must use the REJECTED DATA *reject_file* and EXCEPTIONS *except_file* parameters explicitly. The *reject_file* and *except_file* paths must reside on the client. If *path* resolves to a storage location and the user invoking COPY is not a superuser, the following permissions are required:

- The storage location must have been created with the USER usage type (see [CREATE LOCATION](#)).
- The user must already have been granted access to the storage location where the files exist, as described in [GRANT \(Storage Location\)](#)

Enforce Truncating or Rejecting Rows (ENFORCELENGTH)

When parsing data of type CHAR, VARCHAR, BINARY, or VARBINARY, rows may exceed the target table length. By default, COPY truncates such rows without rejecting them.

Use the ENFORCELENGTH parameter to reject rows that exceed the target table.

For example, loading 'abc' into a table column specified as VARCHAR(2) results in COPY truncating the value to 'ab' and loading it. Loading the same row with the ENFORCELENGTH parameter causes COPY to reject the row.

Note: Vertica supports NATIVE and NATIVE VARCHAR values up to 65K. If any value exceeds this limit, COPY rejects the row, even when ENFORCELENGTH is not in use.

Specify a Maximum Number of Rejections (REJECTMAX)

The REJECTMAX parameter specifies the maximum number of logical records that can be rejected before a load fails. A rejected row consists of the data that could not be parsed into the corresponding data type during a bulk load. Rejected data does not indicate referential constraints. For information about using constraints, and the option of enforcing constraints during bulk loading, see [About Constraints](#).

When the number of rejected records becomes equal to the REJECTMAX value, the load fails. If you do not specify a value for REJECTMAX, or if the value is 0, COPY allows an unlimited number of exceptions to occur.

Abort Data Loads for Any Error (ABORT ON ERROR)

Using the ABORT ON ERROR argument is the most restrictive way to load data, because no exceptions or rejections are allowed. A COPY operation stops if any row is rejected. No data is loaded and Vertica rolls back the command.

If you use the ABORT ON ERROR as part of a CREATE EXTERNAL TABLE AS COPY FROM statement, the option is used whenever a query references the external table. The offending error is saved in the COPY exceptions or rejected data file.

Understanding Row Rejections and Roll Back Errors

Depending on the type of error that COPY encounters, Vertica does one of the following:

- Rejects the offending row and loads other rows into a table
- Rolls back the entire COPY statement without loading any data

Note: If you specify `ABORT ON ERROR` with the COPY statement, the load automatically rolls back if COPY cannot parse *any* row.

The following table summarizes the difference between a rejected row or rollback.

Rejected Rows	Load Roll back
<p>COPY cannot parse rows that contain any of the following:</p> <ul style="list-style-type: none">• Incompatible data types• Missing fields• Missing delimiters	<p>COPY rolls back a statement if it encounters any of these conditions:</p> <ul style="list-style-type: none">• Server-side errors, such as lack of memory• Primary key or foreign key constraint violations• Loading NULL data into a NOT NULL column

This example illustrates what happens when Vertica cannot coerce a row to the requested data type. For example, in the following COPY statement, `"a::INT + b::INT"` is a SQL expression in which a and b are derived values:

```
=> CREATE TABLE t (i INT);
=> COPY t (a FILLER VARCHAR, b FILLER VARCHAR, i AS a::INT + b::INT)
    FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> cat|dog
>> \.
```

Vertica cannot parse the row to the requested data type and rejects the row:

```
ERROR 2827: Could not convert "cat" from column "*FILLER*".a to an int8
```

If a resolved to 'cat' and b to 'dog', the next expression `'cat'::INT + 'dog'::INT` would return an expression evaluator error:

```
=> SELECT 'cat'::INT + 'dog'::INT;
ERROR 3681: Invalid input syntax for integer: "cat"
```

The following COPY statement would also roll back because Vertica cannot parse the row to the requested data type:

```
=> COPY t (a FILLER VARCHAR, i AS a::INT) FROM STDIN;
```

In the following COPY statement, Vertica rejects only the offending row without rolling back the statement. Instead of evaluating the 'cat' row as a VARCHAR type, COPY parses 'cat' directly as an INTEGER.

```
=> COPY t (a FILLER INT, i AS a) FROM STDIN;
```

See Also

- [Cast Failures](#)

Saving Load Rejections (REJECTED DATA)

COPY load rejections are data rows that did not load due to a parser exception. By default, if you do not specify a rejected data file, COPY saves rejected data files to this location:

catalog_dir/CopyErrorLogs/target_table-source-copy-from-rejected-data.n

<i>catalog_dir/</i>	The database catalog files directory, for example: <code>/home/dbadmin/VMart/v_vmart_node0001_catalog</code>
<i>target_table</i>	The table into which data was loaded (<i>target_table</i>).
<i>source</i>	The source of the load data, which can be STDIN, or a file name, such as <code>baseball.csv</code> .
<i>copy-from-rejected-data.n</i>	The default name for a rejected data file, followed by <i>n</i> suffix, indicating the number of files, such as <code>.1</code> , <code>.2</code> , <code>.3</code> . For example, this default file name indicates file 3 after loading from STDIN: <code>fw-STDIN-copy-from-rejected-data.3</code> .

Saving rejected data to the default location, or to a location of your choice, lets you review the file contents, resolve problems, and reload the data from the rejected data files. Saving rejected data to a table, lets you query the table to see rejected data rows and the reasons (exceptions) why the rows could not be parsed. Vertica recommends saving rejected data to a table.

Multiple Rejected Data Files

Unless a load is very small (< 10MB), COPY creates more than one file to hold rejected rows. Several factors determine how many files COPY creates for rejected data. Here are some of the factors:

- Number of sources being loaded
- Total number of rejected rows
- Size of the source file (or files)
- Cooperative parsing and number of threads being used
- UDLs that support apportioned loads
- For your own COPY parser, the number of objects returned from `prepareUDSources()`

Naming Conventions for Rejected Files

You can specify one or more rejected data files with the files you are loading. Use the REJECTED DATA parameter to specify a file location and name, and separate consecutive rejected data file names with a comma (,). Do not use the ON ANY NODE option because it is applicable only to load files.

If you specify one or more files, and COPY requires multiple files for rejected data, COPY uses the rejected data file names you supply as a prefix, and appends a numeric suffix to each rejected data file. For example, if you specify the name `my_rejects` for the REJECTED_DATA parameter, and the file you are loading is large enough (> 10MB), several files such as the following will exist:

- `my_rejects-1`
- `my_rejects-2`
- `my_rejects-3`

COPY uses cooperative parsing by default, having the nodes parse a specific part of the file contents. Depending on the file or portion size, each thread generates at least one rejected data file per source file or portion, and returns load results to the initiator node. The file suffix is a thread index when COPY uses multiple threads (.1, .2, .3, and so on).

The maximum number of rejected data files cannot be greater than the number of sources being loaded, per thread to parse any portion. The resource pool determines the maximum number of threads. For cooperative parse, use all available threads.

If you use COPY with a UDL that supports apportioned load, the file suffix is an offset value. UDL's that support apportioned loading render cooperative parsing unnecessary. For apportioned loads, COPY creates at least one rejected file per data portion, and more files depending on the size of the load and number of rejected rows.

For all data loads except COPY LOCAL, COPY behaves as follows:

No rejected data file specified...	Rejected data file specified...
For a single data file (<i>pathToData</i> or STDIN), COPY stores one or more rejected data files in the default location.	For one data file, COPY interprets the rejected data path as a file, and stores all rejected data at the location. If more than one files is required from parallel processing, COPY appends a numeric suffix. If the path is not a file, COPY returns an error.
For multiple source files, COPY stores all rejected data in separate files in the default directory, using the source file as a file name prefix, as noted.	For multiple source files, COPY interprets the rejected path as a directory. COPY stores all information in separate files, one for each source. If path is not a directory, COPY returns an error. COPY accepts only one path per node. For example, if you specify the rejected data path as <code>my_rejected_data</code> , COPY creates a directory of that name on each node. If you provide more than one rejected data path, COPY returns an error.
Rejected data files are returned to the initiator node.	Rejected data files are not shipped to the initiator node.

Maximum Length File Names

Loading multiple input files in one statement, requires specifying full path names for each file. Keep in mind that long input file names, combined with rejected data file names, can exceed the operating system's maximum length (typically 255 characters). To work around file names that exceed the maximum length, use a path for the rejected data file that differs from the default path—for example, `\tmp\.`

Saving Rejected Data To a Table

Use the `REJECTED DATA` parameter with the `AS TABLE` clause to specify a table in which to save rejected data. Saving rejected data to a file is mutually exclusive with using the `AS TABLE` clause.

When you use the `AS TABLE` clause, Vertica creates a new table if one does not exist, or appends to an existing table. If no parsing rejections occur during a load, the table exists but is empty. The next time you load data, Vertica inserts any rejected rows to the existing table.

The load rejection tables are a special type of table with the following capabilities and limitations:

- Support `SELECT` statements
- Can use `DROP TABLE`
- Cannot be created outside of a `COPY` statement
- Do not support DML and DDL activities
- Are not K-safe

To make the data in a rejected table K-safe, you can do one of the following:

- Write a `CREATE TABLE . . AS` statement, such as this example:

```
=> CREATE TABLE new_table AS SELECT * FROM rejected_table;
```

- Create a table to store rejected records, and run `INSERT . . SELECT` operations into the new table

Using `COPY NO COMMIT`

If the `COPY` statement includes options `NO COMMIT` and `REJECTED DATA AS TABLE`, and the *reject-table* does not already exist, Vertica Analytics Platform saves the rejected data table as a `LOCAL TEMP` table and returns a message that a `LOCAL TEMP` table is being created.

Rejected-data tables are useful for Extract-Load-Transform workflows, where you will likely use temporary tables more frequently. The rejected-data tables let you quickly load data and identify which records failed to load. If you load data into a temporary table that you created using the `ON COMMIT DELETE` clause, the `COPY` operation will not commit.

Location of Rejected Data Table Records

When you save rejected records to a table, using the `REJECTED DATA AS TABLE table_name` option, the data for the table is saved in a database data subdirectory, `RejectionTableData`. For example, for a VMart database, table data files reside here:

```
/home/dbadmin/VMart/v_vmart_node0001_data/RejectionTableData
```

Rejected data tables include both rejected data and the reason for the rejection (exceptions), along with other data columns, described next. Vertica suggests that you periodically drop any rejected data tables that you no longer require.

Querying a Rejected Data Table

When you specify a rejected data table when loading data with `COPY`, you can query that table for information about rejected data after the load operation is complete. For example:

1. Create the loader table:

```
=> CREATE TABLE loader(a INT)
CREATE TABLE
```

2. Use `COPY` to load values, saving rejected data to a table, `loader_rejects`:

```
=> COPY loader FROM STDIN REJECTED DATA AS TABLE loader_rejects;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> a
>> \.
```

3. Query the loader table after loading data:

```
=> SELECT * FROM loader;
 x
 ---
 1
 2
 3
(3 rows)
```

4. Query the `loader_rejects` table to see its column rows:

```
=> SELECT * FROM loader_rejects;
-[ RECORD 1 ]-----+-----
node_name           | v_vmart_node0001
file_name           | STDIN
session_id          | v_vmart_node0001.example.-24016:0x3439
transaction_id      | 45035996274080923
statement_id        | 1
batch_number        | 0
row_number          | 4
rejected_data       | a
rejected_data_orig_length | 1
rejected_reason     | Invalid integer format 'a' for column 1 (x)
```

The rejected data table has the following columns:

Column	Data Type	Description
node_name	VARCHAR	The name of the Vertica node on which the input load file was located.
file_name	VARCHAR	The name of the file being loaded, which applies if you loaded a file (as opposed to using STDIN).
session_id	VARCHAR	The session ID number in which the COPY statement occurred.
transaction_id	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
statement_id	INTEGER	The unique identification number of the statement within the transaction that included the rejected data. <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Tip: You can use the <code>session_id</code>, <code>transaction_id</code>, and <code>statement_id</code> columns to create joins with many system tables. For example, if you join against the <code>QUERY_REQUESTS</code> table using those three columns, the <code>QUERY_REQUESTS.REQUEST</code> column contains the actual COPY statement (as a string) used to load this data.</p> </div>
batch_number	INTEGER	INTERNAL USE. Represents which batch (chunk) the data comes from.

row_number	INTEGER	The rejected row number from the input file.
rejected_data	LONG VARCHAR	The data that was not loaded.
rejected_data_orig_length	INTEGER	The length of the rejected data.
rejected_reason	VARCHAR	The error that caused the rejected row. This column returns the same message that exists in a load exceptions file when you do not save to a table.

Exporting the Rejected Records Table

You can export the contents of the column `rejected_data` to a file to capture only the data rejected during the first `COPY` statement. Then, correct the data in the file, save it, and load the updated file.

To export rejected records:

1. Create a sample table:

```
=> CREATE TABLE t (i int);  
CREATE TABLE
```

2. Copy data directly into the table, using a table to store rejected data:

```
=> COPY t FROM STDIN REJECTED DATA AS TABLE t_rejects;  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 1  
>> 2  
>> 3  
>> 4  
>> a  
>> b  
>> c  
>> \.
```

3. Show only tuples and set the output format:

```
=> \t  
Showing only tuples.  
=> \a  
Output format is unaligned.
```

4. Output to a file:

```
=> \o rejected.txt  
=> select rejected_data from t_rejects;  
=> \o
```

5. Use the catcommand on the saved file:

```
=> \! cat rejected.txt  
a  
b  
c
```

After a file exists, you can fix load errors and use the corrected file as load input to the COPY statement.

Saving Load Exceptions (EXCEPTIONS)

COPY exceptions consist of informational messages describing why a row of data could not be parsed. The optional EXCEPTIONS parameter lets you specify a file to which COPY writes exceptions. If you do not use this parameter, COPY saves exception files to the following default location:

catalog_dir/CopyErrorLogs/tablename-filename-of-source-copy-from-exceptions

<i>catalog_dir</i>	The database catalog files directory
<i>tablename-filename- of-source</i>	The names of the table and data file
<i>-copy-from-exceptions</i>	The file suffix appended to the table and source file name

Note: Using REJECTED DATA AS TABLE *r_table* is mutually exclusive with using the EXCEPTIONS *filename* parameter. The rejected data table includes a column with the exceptions messages. COPY does not permit both parameters. Trying to do so results in this error:

```
ERROR 0: Cannot specify both an exceptions file and a rejected table in the same statement
```

The optional EXCEPTIONS parameter lets you specify a file of your choice to which COPY writes load exceptions. The EXCEPTIONS file indicates the input line number and the reason for each data record exception in this format:

```
COPY: Input record number in <pathofinputfile> has been rejected (reason). Please see pathrejectfile, record recordnum for the rejected record.
```

If copying from STDIN, the *filename-of-source* is STDIN.

Note: You can use specific rejected data and exceptions files with one or more of the files you are loading. Separate consecutive rejected data and exception file names with a comma (,) in the COPY statement.

You must specify a filename in the path to load multiple input files. Keep in mind that long table names combined with long data file names can exceed the operating system's maximum length (typically 255 characters). To work around file names exceeding the maximum length, use a path for the exceptions file that differs from the default path; for example, `\tmp\<shorter-file-name>`.

For all data loads (except for COPY LOCAL), COPY behaves as follows:

No Exceptions file specified...	Exceptions file specified...
For one data source file (<i>pathToData</i> or STDIN), all information stored as one file in the default directory.	For one data file, the path is treated as a file, and COPY stores all information in this file. If the path is not a file, COPY returns an error.
For multiple data files, all information stored as separate files, one for each data file in default directory.	For multiple data files, the path is treated as a directory and COPY stores all information in separate files, one for each data file. If path is not a directory, COPY returns an error.
	Exceptions files are not stored on the initiator node.
	You can specify only one path per node. If you specify more than one path per node, COPY returns an error.

COPY Rejected Data and Exception Files

When executing a COPY statement, and parallel processing is ON (the default setting), COPY creates separate threads to process load files. Typically, the number of threads depends on the number of node cores in the system. Each node processes part of the load data. If the load succeeds overall, any parser rejections that occur during load processing are written to that node's specific rejected data and exceptions files. If the load fails, the rejected data file contents can be incomplete, or empty. If you do not specify a file name explicitly, COPY uses a default name and location for rejected data files. See the next topic for specifying your own rejected data and exception files.

Both rejected data and exceptions files are saved and stored on a per-node basis. This example uses multiple files as COPY inputs. Since the statement does not include either the REJECTED

DATA or EXCEPTIONS parameters, rejected data and exceptions files are written to the default location, the database catalog subdirectory, CopyErrorLogs, on each node:

```
\set dir `pwd`/data/ \set remote_dir /vertica/test_dev/tmp_ms/  
\set file1 ''':dir'C1_large_tbl.dat'''  
\set file2 ''':dir'C2_large_tbl.dat'''  
\set file3 ''':remote_dir'C3_large_tbl.dat'''  
\set file4 ''':remote_dir'C4_large_tbl.dat'''  
  
=>COPY large_tbl FROM :file1 ON site01,:file2 ON site01,  
                    :file3 ON site02,  
                    :file4 ON site02  
                    DELIMITER '|';
```

Note: The RETURNREJECTED parameter is supported only for internal use by the JDBC and ODBC drivers. Be aware that the Vertica internal-use options can change without notice.

Specifying Rejected Data and Exceptions Files

The optional COPY REJECTED DATA and EXCEPTIONS parameters 'path' element lets you specify a non-default path in which to store the files.

If *path* resolves to a storage location, and the user invoking COPY is not a superuser, these are the required permissions:

- The storage location must have been created (or altered) with the USER option (see [CREATE LOCATION](#) and [ALTER_LOCATION_USE](#))
- The user must already have been granted READ access to the storage location where the file (s) exist, as described in [GRANT \(Storage Location\)](#)

Both parameters also have an optional ON *nodename* clause that uses the specified path:

```
...[ EXCEPTIONS 'path' [ ON nodename ] [, ... ] ]...[ REJECTED DATA 'path' [ ON nodename ] [, ... ] ]
```

While '*path*' specifies the location of the rejected data and exceptions files (with their corresponding parameters), the optional ON *nodename* clause moves any existing rejected data and exception files on the node to the specified path on the same node.

Saving Rejected Data and Exceptions Files to a Single Server

The COPY statement does not have a facility to merge exception and rejected data files after COPY processing is complete. To see the contents of exception and rejected data files requires accessing each node's specific files.

Note: To save all exceptions and rejected data files on a network host, be sure to give each node's files unique names, so that different cluster nodes do not overwrite other nodes' files. For instance, if you set up a server with two directories (`/vertica/exceptions` and `/vertica/rejections`), specify file names for each Vertica cluster node to identify each node, such as `node01_exceptions.txt` and `node02_exceptions.txt`. This way, each cluster node's files are easily distinguishable in the exceptions and rejections directories.

Using VSQL Variables for Rejected Data and Exceptions Files

This example uses `vsql` variables to specify the path and file names to use with the exceptions and rejected data parameters (`except_s1` and `reject_s1`). The `COPY` statement specifies a single input file (`large_tbl`) on the initiator node:

```
\set dir `pwd`/data/ \set file1 ''':dir'C1_large_tbl.dat''
\set except_s1 ''':dir'exceptions''
\set reject_s1 ''':dir'rejections''

COPY large_tbl FROM :file1 ON site01 DELIMITER '|'
REJECTED DATA :reject_s1 ON site01
EXCEPTIONS :except_s1 ON site01;
```

This example uses variables to specify exception and rejected data files (`except_s2` and `reject_s2`) on a remote node. The `COPY` statement consists of a single input file on a remote node (`site02`):

```
\set remote_dir /vertica/test_dev/tmp_ms/ \set except_s2 ''':remote_dir'exceptions''
\set reject_s2 ''':remote_dir'rejections''

COPY large_tbl FROM :file1 ON site02 DELIMITER '|'
REJECTED DATA :reject_s2 ON site02
EXCEPTIONS :except_s2 ON site02;
```

This example uses variables to specify that the exception and rejected data files are on a remote node (indicated by `:remote_dir`). The inputs to the `COPY` statement consist of multiple data files on two nodes (`site01` and `site02`). The exceptions and rejected data options use the `ON nodename` clause with the variables to indicate where the files reside (`site01` and `site02`):

```
\set dir `pwd`/data/ \set remote_dir /vertica/test_dev/tmp_ms/
\set except_s1 ''':dir''''
\set reject_s1 ''':dir''''
\set except_s2 ''':remote_dir''''
\set reject_s2 ''':remote_dir''''

COPY large_tbl FROM :file1 ON site01,
                  :file2 ON site01,
```

```
:file3 ON site02,  
:file4 ON site02  
DELIMITER '|'   
REJECTED DATA :reject_s1 ON site01, :reject_s2 ON site02  
EXCEPTIONS :except_s1 ON site01, :except_s2 ON site02;
```

COPY LOCAL Rejection and Exception Files

Invoking COPY LOCAL (or COPY LOCAL FROM STDIN) does not automatically create rejected data and exceptions files. This behavior differs from using COPY, which saves both files automatically, regardless of whether you use the optional REJECTED DATA and EXCEPTIONS parameters to specify either file explicitly.

Use the REJECTED DATA and EXCEPTIONS parameters with COPY LOCAL and COPY LOCAL FROM STDIN to save the corresponding output files on the client. If you do *not* use these options, rejected data parsing events (and the exceptions that describe them) are not retained, even if they occur.

You can load multiple input files using COPY LOCAL (or COPY LOCAL FROM STDIN). If you also use the REJECTED DATA and EXCEPTIONS options, the statement writes rejected rows and exceptions and to separate files. The respective files contain all rejected rows and corresponding exceptions, respectively, regardless of how many input files were loaded.

Note: Because COPY LOCAL (and COPY LOCAL FROM STDIN) must write any rejected rows and exceptions to the client, you cannot use the [ON nodename] clause with either the rejected data or exceptions options.

Specifying Rejected Data and Exceptions Files

To save any rejected data and their exceptions to files:

1. In the COPY LOCAL (and COPY LOCAL FROM STDIN) statement, use the REJECTED DATA 'path' and the EXCEPTIONS 'path' parameters, respectively.
2. Specify two different file names for the two options. You cannot use one file for both the REJECTED DATA and the EXCEPTIONS.
3. When you invoke COPY LOCAL or COPY LOCAL FROM STDIN, the files you specify need not pre-exist. If they do, COPY LOCAL must be able to overwrite them.

You can specify the path and file names with vsql variables:

```
\set rejected ../except_reject/copyLocal.rejected  
\set exceptions ../except_reject/copyLocal.exceptions
```

Note: Using COPY LOCAL does not support storing rejected data in a table, as you can when using the COPY statement.

When you use the COPY LOCAL or COPY LOCAL FROM STDIN statement, specify the variable names for the files with their corresponding parameters:

```
=> COPY large_tbl FROM LOCAL rejected data :rejected exceptions :exceptions;  
=> COPY large_tbl FROM LOCAL STDIN rejected data :rejected exceptions :exceptions;
```

Referential Integrity Load Violation

Vertica checks for constraint violations during query execution. It checks for violations during load operations only if you enable primary or unique key enforcement. In this case, Vertica checks for referential integrity constraint violations when loading data.

If Vertica detects a referential integrity constraint violation, the load rolls back and Vertica returns an error.

See Also

- [Detecting Constraint Violations with ANALYZE_CONSTRAINTS](#)
- [COPY](#)
- [ANALYZE_CONSTRAINTS](#)

Trickle Loading Data

After an initial bulk data load is complete, use `COPY . . . TRICKLE` to load data incrementally with new and changed data.

By default, your database uses transaction isolation level `READ COMMITTED`. Using this isolation level has these benefits:

- Users can see the most recently committed data without holding any locks.
- New data can be loaded while concurrent queries are running.

See [Change Transaction Isolation Levels](#).

Using INSERT, UPDATE, and DELETE

DML statements `INSERT`, `UPDATE`, and `DELETE` perform the same functions that they do in any ACID compliant database. `INSERT` is typically used to load individual rows into physical memory or load a table using an `INSERT AS SELECT` clause. `UPDATE` and `DELETE` are used to modify the data.

You can intermix `INSERT`, `UPDATE`, and `DELETE`. Vertica follows the SQL-92 transaction model. In other words, you do not have to explicitly start a transaction but you must use a `COMMIT` or `ROLLBACK` command (or `COPY`) to end a transaction. If you cancel a DML statement, Vertica rolls back the statement.

Vertica differs from traditional databases in two ways:

- `DELETE` does not actually delete data from disk storage; it marks rows as deleted so they are available for historical queries.
- `UPDATE` writes two rows: one with new data and one marked for deletion.

Unless you specify otherwise, `INSERT`, `UPDATE` and `DELETE` statements write data to the WOS. On overflow, the statements write data to the ROS.

Tip: For large `INSERT` or `UPDATE` operations, use the `DIRECT` hint. This hint forces Vertica to load all data directly to ROS. Loading large number of rows as single row inserts is not recommended for performance reasons. Instead, use `COPY`.

Use of WOS for Trickle Loads

During trickle loading, the WOS (node memory) permits the Vertica to efficiently batch small data loads into larger ones for I/O purposes. Loading to WOS is significantly faster than loading to disk, but it has size constraints. To help balance speed, memory size, and data flow, trickle loading defers the work of sorting, encoding, and writing to disk. The Tuple Mover's moveout process performs those tasks in the background when a table or DML operation uses trickle loading as its [load method](#).

WOS capacity is significantly less than ROS. When incoming data exceeds WOS capacity, Vertica spills small loads directly to disk. While no data is lost when the WOS gets full and spills to ROS, it can result in wasted I/O bandwidth. For optimal performance, follow the guidelines described in [Managing the Tuple Mover](#) to avoid WOS overflow.

If the WOS becomes full, before the tuple mover can move data to ROS, the entire load fails and is rolled back.

Importing and Exporting Data Across Databases

Vertica can easily import data from and export data to other Vertica databases. Importing and exporting data is useful for common tasks such as moving data back and forth between a development or test database and a production database, or between databases that have different purposes but need to share data on a regular basis.

Moving Data Directly Between Databases

Use the following statements move data to and from another Vertica database:

- [COPY FROM VERTICA](#)
- [EXPORT TO VERTICA](#)

To execute either of these statements requires first creating a connection to the other Vertica database.

Creating SQL Scripts to Export Data

Three functions return a SQL script you can use to export database objects to recreate elsewhere:

- [EXPORT_CATALOG](#)
- [EXPORT_OBJECTS](#)
- [EXPORT_TABLES](#)

While copying and exporting data is similar to [Backing Up and Restoring the Database](#), you should use them for different purposes, outlined below:

Task	Backup and Restore	COPY and EXPORT Statements
Back up or restore an entire database, or	YES	NO

Task	Backup and Restore	COPY and EXPORT Statements
incremental changes		
Manage database objects (a single table or selected table rows)	YES	YES
Use external locations to back up and restore your database	YES	NO
Use direct connections between two databases	NO	YES
Use external shell scripts to back up and restore your database	YES	NO
Use SQL commands to incorporate copy and export tasks into DB operations	NO	YES

The following sections explain how you import and export data between Vertica databases.

When importing from or exporting to a Vertica database, you can connect only to a database that uses trusted- (username-only) or password-based authentication, as described in [Security and Authentication](#). SSL authentication is not supported.

Other Exports

This section is about exporting data to another Vertica database. For information about exporting data to Parquet files in HDFS, see [Exporting Data](#) in [Integrating with Apache Hadoop](#).

Exporting Data to Another Vertica Database

You can export a table, specific columns in a table, or the results of a [SELECT](#) statement to another Vertica database. The table in the target database receiving the exported data must already exist, and have columns that match (or can be coerced to) the data types of the columns you are exporting.

You can import data from an earlier Vertica release, if the earlier release is a version of the last major release before the target database release.

Exported data is always written in AUTO mode. For details, see [Choosing a Load Method](#).

Export Process

Exporting is a three-step process:

1. **CONNECT** connects to the target database.

Caution: The export operation fails if either side of the connection is a single-node cluster installed to `localhost`, or you do not specify a host name or IP address.

2. **EXPORT TO VERTICA** exports the data. You can export only one table at a time. Use multiple **EXPORT** statements to export multiple tables or the results of multiple **SELECT** statements. All statements use the same connection to the target database.
3. **DISCONNECT** disconnects from the target database when the export operation is complete.

Exporting Identity Columns

You can export tables (or columns) that contain identity and auto-increment values, but the sequence values are not incremented automatically at the target table. You must use **ALTER SEQUENCE** to make updates.

Export identity (and auto-increment) columns as follows:

- If both source and destination tables have an identity column and configuration parameter `CopyFromVerticaWithIdentity` is set to `true` (1), you do not need to list them.
- If source table has an identity column, but target table does not, you must explicitly list the source and target columns.

Caution: Failure to list which identity columns to export can cause an error, because the identity column will be interpreted as missing in the destination table.

By default, **EXPORT TO VERTICA** exports all identity columns. To disable this behavior globally, set the `CopyFromVerticaWithIdentity` configuration parameter.

Exporting GEOMETRY and GEOGRAPHY Data Types

You can export columns containing Vertica [geospatial data types](#). Vertica supports the following export methods for these data types:

Export/Copy Support	Target version: 8.0 SP1 and later	Target version: Previous
Source version: 8.0 SP1 and later	Export/Copy	Export
Source version: Previous	Copy	Not supported

Examples of Exporting Data

The following example demonstrates using the three-step process listed above to export data.

First, open the connection to the other database, then perform a simple export of an entire table to an identical table in the target database.

```
=> CONNECT TO VERTICA testdb USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> EXPORT TO VERTICA testdb.customer_dimension FROM customer_dimension;
Rows Exported
-----
                23416
(1 row)
```

The following statement demonstrates exporting a portion of a table using a simple [SELECT](#) statement.

```
=> EXPORT TO VERTICA testdb.ma_customers AS SELECT customer_key, customer_name, annual_income
FROM customer_dimension WHERE customer_state = 'MA';
Rows Exported
-----
                3429
(1 row)
```

This statement exports several columns from one table to several different columns in the target database table using column lists. Remember that when supplying both a source and destination column list, the number of columns must match.

```
=> EXPORT TO VERTICA testdb.people (name, gender, age) FROM customer_dimension
(customer_name, customer_gender, customer_age);
```

```
Rows Exported
-----
          23416
(1 row)
```

You can also use `EXPORT TO VERTICA` with a `SELECT AT EPOCH LATEST` expression to include data from the latest committed DML transaction.

Disconnect from the database when the export is complete:

```
=> DISCONNECT testdb;
DISCONNECT
```

Note: Closing your session also closes the database connection. However, it is a good practice to explicitly close the connection to the other database, both to free up resources and to prevent issues with other SQL scripts you may run in your session. Always closing the connection prevents potential errors if you run a script in the same session that attempts to open a connection to the same database, since each session can only have one connection to a particular database at a time.

Copying Data from Another Vertica Database

You can import a table or specific columns in a table from another Vertica database. The table receiving the copied data must already exist, and have columns that match (or can be coerced into) the data types of the columns you are copying from the other database. You can import data from an earlier Vertica release, if the earlier release is a version of the last major release before the target database release.

Import Process

Importing is a three-step process:

1. `CONNECT` connects to the database that contains the data to import.

Note: The import operation fails if either side of the connection is a single-node cluster installed to `localhost`, or you do not specify a host name or IP address.

2. [COPY FROM VERTICA](#) imports table data to the target database. To import data from multiple tables, issue multiple `COPY FROM VERTICA` statements using the same connection to the source database.
3. [DISCONNECT](#) disconnects from the source database when the copy operation is complete.

Importing Identity Columns

You can import identity (and auto-increment) columns as follows:

- If both source and destination tables have an identity column and configuration parameter `CopyFromVerticaWithIdentity` is set to true (1), you do not need to list them.
- If source table has an identity column, but target table does not, you must explicitly list the source and target columns.

Caution: Failure to list which identity columns to export can cause an error, because the identity column will be interpreted as missing in the destination table.

After importing the columns, the identity column values do not increment automatically. Use [ALTER SEQUENCE](#) to make updates.

The default behavior for this statement is to import Identity (and Auto-increment) columns by specifying them directly in the source table. To disable this behavior globally, set the `CopyFromVerticaWithIdentity` configuration parameter, described in [Configuration Parameters](#).

Examples

This example demonstrates connecting to another database, copying the contents of an entire table from the source database to an identically-defined table in the current database directly into ROS, and then closing the connection:

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY customer_dimension FROM VERTICA vmart.customer_dimension DIRECT;
  Rows Loaded
-----
          500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

This example demonstrates copying several columns from a table in the source database into a table in the local database:

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY people (name, gender, age) FROM VERTICA
-> vmart.customer_dimension (customer_name, customer_gender,
-> customer_age);
Rows Loaded
-----
      500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

Changing Node Export Addresses

You can change the export address for your Vertica cluster. You might need to do so to export data between clusters in different network subnets.

1. Create a subnet for importing and exporting data between Vertica clusters. The CREATE SUBNET statement identifies the public network IP addresses residing on the same subnet.

```
=> CREATE SUBNET kv_subnet with '10.10.10.0';
```

2. Alter the database to specify the subnet name of a public network for import/export.

```
=> ALTER DATABASE VMartDB EXPORT ON kv_subnet;
```

3. Create network interfaces for importing and exporting data from individual nodes to other Vertica clusters. The CREATE NETWORK INTERFACE statement identifies the public network IP addresses residing on multiple subnets.

```
=> CREATE NETWORK INTERFACE kv_node1 on v_VMartDB_node0001 with '10.10.10.1';
=> CREATE NETWORK INTERFACE kv_node2 on v_VMartDB_node0002 with '10.10.10.2';
=> CREATE NETWORK INTERFACE kv_node3 on v_VMartDB_node0003 with '10.10.10.3';
=> CREATE NETWORK INTERFACE kv_node4 on v_VMartDB_node0004 with '10.10.10.4';
```

For users on Amazon Web Services (AWS) or using Network Address Translation (NAT), refer to [Vertica on Amazon Web Services](#).

4. Alter the node settings to change the export address. When used with the EXPORT ON clause, the ALTER NODE specifies the network interface of the public network on individual

nodes for importing and exporting data.

```
=> ALTER NODE v_VMartDB_node0001 export on kv_node1;  
=> ALTER NODE v_VMartDB_node0002 export on kv_node2;  
=> ALTER NODE v_VMartDB_node0003 export on kv_node3;  
=> ALTER NODE v_VMartDB_node0004 export on kv_node4;
```

5. Verify if the node address and the export address are different on different network subnets of the Vertica cluster.

```
=> SELECT node_name, node_address, export_address FROM nodes;  
   node_name      | node_address      | export_address  
-----+-----+-----  
v_VMartDB_node0001 | 192.168.100.101 | 10.10.10.1  
v_VMartDB_node0002 | 192.168.100.102 | 10.10.10.2  
v_VMartDB_node0003 | 192.168.100.103 | 10.10.10.3  
v_VMartDB_node0004 | 192.168.100.104 | 10.10.10.4
```

Creating a network interface and altering the node settings to change the export address takes precedence over creating a subnet and altering the database for import/export.

Using Public and Private IP Networks

In many configurations, Vertica cluster hosts use two network IP addresses as follows:

- A private address for communication between the cluster hosts.
- A public IP address for communication with client connections.

By default, importing from and exporting to another Vertica database uses the private network.

Note: Ensure port 5433 or the port the Vertica database is using is not blocked.

To use the public network address for copy and export activities, as well as moving large amounts of data, configure the system to use the public network to support exporting to or importing from another Vertica cluster:

- [Identify the Public Network to Vertica](#)
- [Identify the Database or Nodes Used for Import/Export](#)

In certain instances, both public and private addresses exceed the demand capacity of a single Local Area Network (LAN). If you encounter this type of scenario, then configure your Vertica cluster to use two LANs: one for public network traffic and one for private network traffic.

Identify the Public Network to Vertica

To be able to import to or export from a public network, Vertica needs to be aware of the IP addresses of the nodes or clusters on the public network that will be used for import/export activities. Your public network might be configured in either of these ways:

- Public network IP addresses reside on the same subnet (create a subnet)
- Public network IP addresses are on multiple subnets (create a network interface)

To identify public network IP addresses residing on the same subnet:

- Use the [CREATE SUBNET](#) statement provide your subnet with a name and to identify the subnet routing prefix.

To identify public network IP addresses residing on multiple subnets:

- Use the [CREATE NETWORK INTERFACE](#) statement to configure import/export from specific nodes in the Vertica cluster.

After you've identified the subnet or network interface to be used for import/export, you must [Identify the Database Or Nodes Used For Import/Export](#).

See Also

- [CREATE SUBNET](#)
- [ALTER SUBNET](#)
- [DROP SUBNET](#)
- [CREATE NETWORK INTERFACE](#)
- [ALTER NETWORK INTERFACE](#)
- [DROP NETWORK INTERFACE](#)

Identify the Database or Nodes Used for Import/Export

After you identify the public network to Vertica, you can configure a database and its nodes to use it for import and export operations:

- Use [ALTER DATABASE](#) to specify a subnet on the public network for the database. After doing so,, all nodes in the database automatically use the network interface on the subnet for import/export operations.
- On each database node, use [ALTER NODE](#) to specify a network interface of the public network.

See Also

- [CREATE PROCEDURE](#)
- [CREATE NETWORK INTERFACE](#)
- [V_MONITOR.NETWORKINTERFACES](#)

Handling Node Failure During Copy/Export

When an export (`EXPORT TO VERTICA`) or import from Vertica (`COPY FROM VERTICA`) task is in progress, and a non-initiator node fails, Vertica does not complete the task automatically. A non-initiator node is any node that is not the source or target node in your export or import statement. To complete the task, you must run the statement again.

You address the problem of a non-initiator node failing during an import or export as follows:

Note: Both Vertica databases must be running in a safe state.

1. You export or import from one cluster to another using the `EXPORT TO VERTICA` or `COPY FROM VERTICA` statement.

During the export or import, a non-initiating node on the target or source cluster fails. Vertica issues an error message that indicates possible node failure, one of the following:

- `ERROR 4534: Receive on v_tpchdb1_node0002: Message receipt from v_tpchdb2_node0005 failed`
 - `WARNING 4539: Received no response from v_tpchdb1_node0004 in abandon plan`
 - `ERROR 3322: [tpchdb2] Execution canceled by operator`
2. Complete your import or export by running the statement again. The failed node does not need to be up for Vertica to successfully complete the export or import.

Using EXPORT Functions

Vertica provides several `EXPORT_` functions that let you recreate a database, or specific schemas and tables, in a target database. For example, you can use the `EXPORT_` functions to transfer some or all of the designs and objects you create in a development or test environment to a production database.

The `EXPORT_` functions create SQL scripts that you can run to generate the exported database designs or objects. These functions serve different purposes to the export statements, [COPY FROM VERTICA](#) (pull data) and [EXPORT TO VERTICA](#) (push data). These statements transfer data directly from source to target database across a network connection between both. They are dynamic actions and do not generate SQL scripts.

The `EXPORT_` functions appear in the following table. Depending on what you need to export, you can use one or more of the functions. `EXPORT_CATALOG` creates the most comprehensive SQL script, while `EXPORT_TABLES` and `EXPORT_OBJECTS` are subsets of that function to narrow the export scope.

Use this function...	To recreate...
EXPORT_CATALOG	These catalog items: <ul style="list-style-type: none">• An existing schema design, tables, projections, constraints, and views• The Database Designer-created schema design, tables, projections, constraints, and views• A design on a different cluster.
EXPORT_TABLES	Non-virtual objects up to, and including, the schema of one or more tables.
EXPORT_OBJECTS	Catalog objects in order dependency for replication.

The designs and object definitions that the script creates depend on the `EXPORT_` function scope you specify. The following sections give examples of the commands and output for each function and the scopes it supports.

Saving Scripts for Export Functions

All of the examples in this section were generated using the standard Vertica VMART database, with some additional test objects and tables. One output directory was created for all SQL scripts that the functions created:

```
/home/dbadmin/xtest
```

If you specify the destination argument as an empty string (' '), the function writes the export results to STDOUT.

Note: A superuser can export all available database output to a file with the EXPORT_ functions. For a non-superuser, the EXPORT_ functions generate a script containing only the objects to which the user has access.

Exporting the Catalog

The function [EXPORT_CATALOG](#) generates a SQL script for copying a database design to another cluster. This script replicates the physical schema design of the source database. You call this function as follows:

```
EXPORT_CATALOG ( [ 'destination' ] , [ 'scope' ] )
```

Note: This function and [EXPORT_OBJECTS](#) return equivalent output.

Setting Scope of Export

You can set the scope of the export operation to various levels:

To export...	Use this scope...
Schemas, tables, constraints, views, and projections	DESIGN (default)
All design objects and system objects created in Database Designer, such as design contexts and their tables	DESIGN ALL
Only tables, constraints, and projection	TABLES

Exporting All Catalog Objects

Use the DESIGN scope to export all design elements of a source database in order dependency. This scope exports all catalog objects in their OID (unique object ID) order, including schemas, tables, constraints, views, and all types of projections. This is the most comprehensive export scope, without the Database Designer elements, if they exist.

```
=> SELECT EXPORT_CATALOG(  
      '/home/dbadmin/xtest/sql_cat_design.sql',  
      'DESIGN' );  
      EXPORT_CATALOG  
-----  
Catalog data exported successfully  
(1 row)
```

The SQL script includes the following types of statements, each needed to recreate a new database:

- CREATE SCHEMA
- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROJECTION (with ORDER BY and SEGMENTED BY)
- ALTER TABLE (to add constraints)
- PARTITION BY

Projection Considerations

If a projection to export was created with no ORDER BY clause, the SQL script reflects the default behavior for projections. Vertica implicitly creates projections using a sort order based on the SELECT columns in the projection definition. The EXPORT_CATALOG script reflects this behavior.

The EXPORT_CATALOG script is portable if all projections were generated using UNSEGMENTED ALL NODES or SEGMENTED ALL NODES.

Exporting Database Designer Schema and Designs

Use the DESIGN ALL scope to generate a script to recreate all design elements of a source database and the design and system objects that were created by the Database Designer:

```
=> SELECT EXPORT_CATALOG (  
    '/home/dbadmin/xtest/sql_cat_design_all.sql',  
    'DESIGN_ALL');  
    EXPORT_CATALOG  
-----  
Catalog data exported successfully  
(1 row)
```

Exporting Table Objects

Use the TABLES scope to generate a script to recreate all schemas tables, constraints, and sequences:

```
=> SELECT EXPORT_CATALOG (  
    '/home/dbadmin/xtest/sql_cat_tables.sql',  
    'TABLES');  
    EXPORT_CATALOG  
-----  
Catalog data exported successfully  
(1 row)
```

The SQL script includes the following types of statements:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE

See Also

- [EXPORT_CATALOG](#)
- [EXPORT_OBJECTS](#)
- [EXPORT_TABLES](#)

- [Exporting Tables](#)
- [Exporting Objects](#)

Exporting Tables

Use the Vertica function [EXPORT_TABLES](#) to recreate one or more tables, and related non-virtual objects, on a different cluster. `EXPORT_TABLES` has the following syntax:

```
EXPORT_TABLES( [ 'destination' ] [, 'scope' ] )
```

Setting Scope of Export

You can set the scope of the export operation to various levels:

To export...	Use this scope...
All objects to which the user has access, including constraints.	An empty string (' ')
One or more named objects, such as tables or sequences in one or more schemas. You can optionally qualify the schema with a database prefix, <code>myvertica.myschema.newtable</code> .	A comma-delimited list of table objects. For example: <code>'myschema.newtable, yourschema.oldtable'</code>
A named table object in the current search path. You can specify a schema, table, or sequence. If you specify a schema, <code>EXPORT_TABLES</code> exports all table objects in that schema to which the user has access.	The table object's name and, optionally, its path: <code>'VMart.myschema'</code>

The SQL script includes only the non-virtual objects to which the current user has access.

Note: `EXPORT_TABLES` does not export views. If you specify a view name, Vertica silently ignores it and the view is omitted from the generated script. To export views, use [EXPORT_OBJECTS](#).

Exporting All Table Objects

If you set the scope parameter to an empty string (' '), Vertica exports all tables and their related objects:

```
=> SELECT EXPORT_TABLES(  
    '/home/dbadmin/xtest/sql_tables_empty.sql',  
    '');  
    EXPORT_TABLES  
-----  
Catalog data exported successfully  
(1 row)
```

The SQL script includes the following types of statements, depending on what is required to recreate the tables and any related objects (such as sequences):

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE
- PARTITION BY

Exporting a List of Tables

Use EXPORT_TABLE with a comma-separated list of objects, including tables, views, or schemas:

```
=> SELECT EXPORT_TABLES(  
    '/home/dbadmin/xtest/sql_tables_del.sql'  
    'public.student, public.test7');  
    EXPORT_TABLES  
-----  
Catalog data exported successfully  
(1 row)
```

The SQL script can include the following types of statements, depending on what is required to recreate the exported objects:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE

Exporting a Single Table Object

Use `EXPORT_TABLES` to export one or more database table objects.

This example exports a named sequence, `my_seq`, qualifying the sequence with the schema name (`public`):

```
=> SELECT EXPORT_TABLES(  
      '/home/dbadmin/xtest/export_one_sequence.sql',  
      'public.my_seq');  
      EXPORT_TABLES  
-----  
Catalog data exported successfully  
(1 row)
```

Following are the contents of the `export_one_sequence.sql` output file using a more command:

```
$ more export_one_sequence.sql  
CREATE SEQUENCE public.my_seq ;
```

See Also

[Exporting Objects](#)

Exporting Objects

The Vertica function `EXPORT_OBJECTS` generates a SQL script that you can use to recreate non-virtual catalog objects on a different cluster, as follows:

```
EXPORT_OBJECTS( [ 'destination' ] [, 'scope' ] [, 'ksafe' ] )
```

Note: This function and `EXPORT_CATALOG` return equivalent output.

Setting Scope of Export

You can set the scope of the export operation to various levels:

To export...	Use this scope...
All non-virtual objects to which the user has access, including constraints.	An empty string (' ')

To export...	Use this scope...
One or more named objects, such as tables or views in one or more schemas. You can optionally qualify the schema with a database prefix, <code>myvertica.myschema.newtable</code> .	A comma-delimited list of items. For example: <code>'myschema.newtable, yourschema.oldtable'</code>
A named database object in the current search path. You can specify a schema, table, or view. If the object is a schema, the script includes objects to which the user has access.	The table object's name and, optionally, its path: <code>'VMart.myschema'</code>

The SQL script includes only the non-virtual objects to which the current user has access.

`EXPORT_OBJECTS` always tries to recreate projection statements with their `KSAFE` clauses, if any; otherwise, with their `OFFSET` clauses.

Function Syntax

```
EXPORT_OBJECTS( [ 'destination' ] , [ 'scope' ] , [ 'ksafe' ] )
```

Exporting All Objects

If you set the `scope` parameter to an empty string (`' '`), Vertica exports all non-virtual objects from the source database in order dependency. Running the generated SQL script on another cluster creates all referenced objects and their dependent objects.

By default, the function's `KSAFE` argument is set to `true`. In this case, the generated script calls `MARK_DESIGN_KSAFE`, which propagates the K-safety setting of the original database.

```
=> SELECT EXPORT_OBJECTS(
      '/home/dbadmin/xtest/sql_objects_all.sql',
      '',
      'true');
      EXPORT_OBJECTS
-----
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements:

- CREATE SCHEMA
- CREATE TABLE

- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROJECTION (with ORDER BY and SEGMENTED BY)
- ALTER TABLE (to add constraints)
- PARTITION BY

Here is a snippet that includes the start and end of the output SQL file, including the MARK_DESIGN_KSAFE statement:

```
CREATE SCHEMA store;
CREATE SCHEMA online_sales;
CREATE SEQUENCE public.my_seq ;
CREATE TABLE public.customer_dimension
(
    customer_key int NOT NULL,
    customer_type varchar(16),
    customer_name varchar(256),
    customer_gender varchar(8),
    title varchar(8),
    household_id int,
    ...
);
.
.
.
SELECT MARK_DESIGN_KSAFE(1);
```

Exporting a List of Objects

Use a comma-separated list of objects as the function scope. The list can include one or more tables, sequences, and views in the same, or different schemas, depending on how you qualify the object name. For instance, specify a table from one schema, and a view from another (schema2.view1).

The SQL script includes the following types of statements, depending on what objects you include in the list:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)

- CREATE VIEW
- CREATE SEQUENCE

If you specify a view without its dependencies, the function displays a WARNING. The SQL script includes a CREATE statement for the dependent object, but will be unable to create it without the necessary relations:

```
=> SELECT EXPORT_OBJECTS(  
      'nameObjectsList',  
      'test2, tt, my_seq, v2' );  
WARNING 0: View public.v2 depends on other relations  
      EXPORT_OBJECTS  
-----  
      Catalog data exported successfully  
(1 row)
```

This example explicitly sets the *ksafe* argument explicitly to true.

```
=> SELECT EXPORT_OBJECTS(  
      '/home/dbadmin/xtest/sql_objects_table_view_KSAFE.sql',  
      'v1, test7',  
      'true');  
      EXPORT_OBJECTS  
-----  
      Catalog data exported successfully  
(1 row)
```

Here are the contents of the output file of the example, showing the sample table test7 and the v1 view:

```
CREATE TABLE public.test7  
(  
  a int,  
  c int NOT NULL DEFAULT 4,  
  bb int  
);  
CREATE VIEW public.v1 AS  
  SELECT tt.a  
  FROM public.tt;  
  
SELECT MARK_DESIGN_KSAFE(1);
```

Exporting a Single Object

Specify a single database object as the function scope. The object can be a schema, table, sequence, or view. The function exports all non-virtual objects associated with the one you specify.

```
=> SELECT EXPORT_OBJECTS(  
      '/home/dbadmin/xtest/sql_objects_viewobject_KSAFE.sql',  
      'v1',  
      'true');  
      EXPORT_OBJECTS  
-----  
Catalog data exported successfully  
(1 row)
```

The output file contains the v1 view:

```
CREATE VIEW public.v1 AS  
  SELECT tt.a  
  FROM public.tt;  
  
SELECT MARK_DESIGN_KSAFE(1);
```

See Also

[Exporting Tables](#)

Managing Storage Locations

Vertica *storage locations* are the paths to file destinations you designate to store data and temporary files. Every node in the cluster requires at least one area in which to store data and another separate area in which to store database catalog files. You set up these locations as part of installation and setup. (See [Prepare Disk Storage Locations](#) in Installing Vertica for disk space requirements.)

Important: While no technical issue prevents you from using `CREATE LOCATION` to add one or more Network File System (NFS) storage locations, Vertica does not support NFS data or catalog storage except for MapR mount points. You will be unable to run queries against any other NFS data. When creating locations on MapR file systems, you must specify `ALL NODES SHARED`.

How Vertica Uses Storage Locations

Every time you add data to the database or perform a DML operation, the new data is held in memory (WOS) and moved to storage locations on disk (ROS) at regular intervals. Depending on the configuration of your database, many ROS containers are likely to exist.

You can label the storage locations you create to reference them in object storage policies. If an object to store has no associated storage policy, Vertica uses the available storage location and default storage algorithms to store data. If the object has a storage policy, Vertica stores the data at the object's designated storage location. See [Creating Storage Policies](#). If you no longer need a storage location, you can retire or drop it, as described in [Retiring Storage Locations](#) and [Dropping Storage Locations](#).

Comparing Local and Shared Storage Locations

You can define two types of storage locations:

- **Local**—Stores data in a unique location for each node. Local is the default for a storage location. You do not have to specify it explicitly. This location is in a directory in a file system that the node can access, and is often in the node's own file system. You can create a local

storage location for a single node or for all nodes in the cluster. Cluster-wide storage locations are the most common type of storage. Vertica defaults to using a local cluster-wide storage location for storing all data. If you want it to store data differently, you must create additional storage locations.

- **Shared**—Stores data on a single file system to which all nodes in the cluster have access. This shared file system is often hosted outside of the cluster, such as on a distributed file system like HDFS. When you create a shared storage location, each node in the Vertica cluster creates its own subdirectory in the location. The separate directories prevent nodes from overwriting each other's data. Currently, Vertica supports only HDFS shared storage locations. You cannot use NFS as a Vertica shared storage location except when using MapR mount points. See [Vertica Storage Location for HDFS](#) in Integrating with Apache Hadoop for more information.

Viewing Storage Locations and Policies

You can monitor information about available storage location labels and your current storage policies.

Viewing Disk Storage Information

Query the [V_MONITOR.DISK_STORAGE](#) system table for disk storage information on each database node. For more information, see [Using System Tables](#) and [Altering Location Use](#). The [V_MONITOR.DISK_STORAGE](#) system table includes a CATALOG annotation, indicating that the location is used to store catalog files.

Note: You cannot add or remove a catalog storage location. Vertica creates and manages this storage location internally, and the area exists in the same location on each cluster node.

Viewing Location Labels

Three system tables have information about storage location labels in their `location_labels` columns:

- `storage_containers`
- `storage_locations`
- `partitions`

Use a query such as the following for relevant columns of the `storage_containers` system table:

```
VMART=> select node_name,projection_name, location_label from v_monitor.storage_containers;
node_name      | projection_name | location_label
-----+-----+-----
v_vmart_node0001 | states_p       |
v_vmart_node0001 | states_p       |
v_vmart_node0001 | t1_b1          |
v_vmart_node0001 | newstates_b0  | FAST3
v_vmart_node0001 | newstates_b0  | FAST3
v_vmart_node0001 | newstates_b1  | FAST3
v_vmart_node0001 | newstates_b1  | FAST3
v_vmart_node0001 | newstates_b1  | FAST3
.
.
.
```

Use a query such as the following for columns of the `v_catalog.storage_locations` system table:

```
VMart=> select node_name, location_path, location_usage, location_label from storage_locations;
node_name      | location_path   | location_usage | location_label
-----+-----+-----+-----
v_vmart_node0001 | /home/dbadmin/VMart/v_vmart_node0001_data | DATA,TEMP    |
v_vmart_node0001 | home/dbadmin/SSD/schemas | DATA         |
v_vmart_node0001 | /home/dbadmin/SSD/tables | DATA         | SSD
v_vmart_node0001 | /home/dbadmin/SSD/schemas | DATA         | Schema
v_vmart_node0002 | /home/dbadmin/VMart/v_vmart_node0002_data | DATA,TEMP    |
v_vmart_node0002 | /home/dbadmin/SSD/tables | DATA         |
v_vmart_node0002 | /home/dbadmin/SSD/schemas | DATA         |
v_vmart_node0003 | /home/dbadmin/VMart/v_vmart_node0003_data | DATA,TEMP    |
v_vmart_node0003 | /home/dbadmin/SSD/tables | DATA         |
v_vmart_node0003 | /home/dbadmin/SSD/schemas | DATA         |
(10 rows)
```

Use a query such as the following for columns of the `v_monitor.partitions` system table:

```
VMART=> select partition_key, projection_name, location_label from v_monitor.partitions;
partition_key | projection_name | location_label
-----+-----+-----
NH            | states_b0      | FAST3
MA            | states_b0      | FAST3
VT            | states_b1      | FAST3
ME            | states_b1      | FAST3
CT            | states_b1      | FAST3
.
.
```

Viewing Storage Tiers

Query the `storage_tiers` system table to see both the labeled and unlabeled storage containers and information about them:

```
VMart=> select * from v_monitor.storage_tiers;
location_label | node_count | location_count | ros_container_count | total_occupied_size
-----+-----+-----+-----+-----
SSD            |          1 |              2 |                   17 |          297039391
Schema        |          1 |              1 |                   9 |           1506
(3 rows)
```

Viewing Storage Policies

Query the `storage_policies` system table to view the current storage policy in place.

```
VMART=> select * from v_monitor.storage_policies;
schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public      | public     | Schema        | F4
public      | lineorder  | Partition [4, 4] | M3
(2 rows)
```

Creating Storage Locations

You can add and configure storage locations (other than the required defaults) to provide storage for these purposes:

- Isolating execution engine temporary files from data files.
- Creating labeled locations to use in storage policies.
- Creating storage locations based on predicted or measured access patterns.
- Creating USER storage locations for specific users or user groups.

Important: While no technical issue prevents you from using `CREATE LOCATION` to add one or more Network File System (NFS) storage locations, Vertica does not support

NFS data or catalog storage except for MapR mount points. You will be unable to run queries against any other NFS data. When creating locations on MapR file systems, you must specify `ALL NODES SHARED`.

You can add a new storage location from one node to another node or from a single node to all cluster nodes. However, do not use a shared directory on one node for other cluster nodes to access.

Planning Storage Locations

Adding a storage location requires minimal planning:

1. Verify that the directory you plan to use for a storage location destination is an empty directory with write permissions for the Vertica process.
2. Plan the labels to use if you want to label the location as you create it.
3. Determine the type of information to store in the storage location:
 - `DATA` — Persistent data and temp table data.
 - `TEMP` — Temporary files that are generated and dumped to disk, such as those generated by sort, group by, join, and so on.
 - `DATA,TEMP` — Both data and temp files (the default).
 - `USER` — Gives access to non-dbadmin users so they can use the storage location after being granted read or write privileges. You cannot assign this location type for use in a storage policy.

Tip: Storing temp and data files in different storage locations is advantageous because the two types of data have different disk I/O access patterns. Temp files are distributed across locations based on available storage space. However, data files can be stored on different storage locations, based on storage policy, to reflect predicted or measured access patterns.

If you plan to place storage locations on HDFS, see [Using HDFS Storage Locations](#) in Integrating with Apache Hadoop for additional requirements.

Creating Unlabeled Local Storage Locations

This example shows a three-node cluster, each with a `vertica/SSD` directory for storage.



On each node in the cluster, create a directory where the node stores its data. For example:

```
$ mkdir /home/dbadmin/vertica/SSD
```

Micro Focus recommends that you create the same directory path on each node. Use this path when creating a storage location.

Use the [CREATE LOCATION](#) statement to add a storage location. Specify the following information:

- The path on the node where Vertica stores the data.
- The node where the location is available, or `ALL NODES`.

Important: Specify the node or use the `ALL NODES` keyword. Otherwise, the statement creates the storage locations on all nodes in the cluster in a single transaction.

- The type of information to be stored.

For user access (non-dbadmin users), you must create the storage location with the `USER` usage type. You cannot change an existing storage location to have `USER` access. After you create a `USER` storage location, you can grant one or more users access to it. User areas can store only data files, not temp files. You cannot assign a `USER` storage location to a storage policy.

The following example shows how to add a location available on all nodes to store only data:

```
CREATE LOCATION '/home/dbadmin/vertica/SSD/' ALL NODES USAGE 'DATA';
```

The following example shows how to add a location that is available on node `v_vmart_node0001` to store data and temporary files:

```
CREATE LOCATION '/home/dbadmin/vertica/SSD/' NODE 'v_vmart_node0001';
```

Suppose you are using a storage location for data files and want to create ranked storage locations. In this ranking, columns are stored on different disks based on their measured performance. To create ranked storage locations, see the following sections:

1. [Measuring Storage Performance](#)
2. [Setting Storage Performance](#)

Note: After you create a storage location, you can alter the type of information it stores, with some restrictions. See [Altering Location Use](#).

Storage Location Subdirectories

You cannot create a storage location in a subdirectory of an existing location. Doing so results in an error similar to the following:

```
=> CREATE LOCATION '/tmp/myloc' ALL NODES USAGE 'TEMP';  
CREATE LOCATION  
=> CREATE LOCATION '/tmp/myloc/ssd' ALL NODES USAGE 'TEMP';  
ERROR 5615: Location [/tmp/myloc/ssd] conflicts with existing location  
[/tmp/myloc] on node v_vmart_node0001
```

Creating Labeled Storage Locations

You can add a storage location with a descriptive label using the `CREATE LOCATION` statement's `LABEL` keyword. You use labeled locations to set up storage policies for your site. See [Creating Storage Policies](#).

This example shows how to create a storage location on `v_vmart_node0002` with the label `SSD`:

```
=> CREATE LOCATION '/home/dbadmin/SSD/schemas' NODE 'v_vmart_node0002'  
USAGE 'DATA' LABEL 'SSD';
```

This example shows you how to create a storage location on all nodes. Specifying the `ALL NODES` keyword adds the storage location to all nodes in a single transaction:

```
=> CREATE LOCATION '/home/dbadmin/SSD/schemas' ALL NODES  
    USAGE 'DATA' LABEL 'SSD';
```

The new storage location is listed in the `v_monitor.disk_storage` system table:

```
=> SELECT * FROM v_monitor.disk_storage;  
. .  
-[ RECORD 7 ]-----+-----  
node_name          | v_vmart_node0002  
storage_path       | /home/dbadmin/SSD/schemas  
storage_usage      | DATA  
rank               | 0  
throughput         | 0  
latency            | 0  
storage_status     | Active  
disk_block_size_bytes | 4096  
disk_space_used_blocks | 1549437  
disk_space_used_mb  | 6053  
disk_space_free_blocks | 13380004  
disk_space_free_mb  | 52265  
disk_space_free_percent | 89%  
. . .
```

Creating a Storage Location for USER Access

You can create USER storage locations for a non-dbadmin user to access the storage location after being granted appropriate privileges.

The following example shows how to create a storage location, `BillyBobStore`, with a USER usage argument, on node `v_mcdb_node0007`:

```
=> CREATE LOCATION '/home/dbadmin/UserStorage/BillyBobStore' NODE  
    'v_mcdb_node0007' USAGE 'USER';
```

The following example shows how to grant a user `BillyBob` read and write permissions to the `BillyBobStore` location:

```
=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BillyBobStore' TO BillyBob;  
GRANT PRIVILEGE
```

For more information about configuring user privileges, see [Managing Users and Privileges](#) in the Administrator's Guide and the [GRANT \(Storage Location\)](#) and [REVOKE \(Storage Location\)](#) functions in the SQL Reference Manual.

Altering Location Use

You can make changes to the type of files that Vertica stores at a storage location. To modify a storage location, use the [ALTER_LOCATION_USE](#) function. Normally you use labels only for DATA storage locations, not TEMP.

This example shows how to alter the storage location on `v_vmartdb_node0004` to store only data files:

```
=> SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/' , 'v_vmartdb_node0004' , 'DATA');
```

Altering HDFS Storage Locations

When altering an HDFS storage location, you must make the change for each node in the Vertica cluster. You can make the change on all nodes at the same time by specifying a node value of "", as in the following example:

```
=> SELECT ALTER_LOCATION_USE('webhdfs://hadoop:50070/user/dbadmin/v_vmart',  
    '', 'TEMP');
```

USER Storage Location Restrictions

You cannot change a storage location from a USER usage type if you created the location that way, or to a USER type if you did not. You can change a USER storage location to specify DATA (storing TEMP files is not supported). However, doing so does not affect the primary objective of a USER storage location, to be accessible by non-dbadmin users with assigned privileges.

Effects of Altering Storage Location Use

Before altering a storage location use type, be aware that at least one location must remain for storing data and temp files on a node. You can store data and temp files in the same, or separate, storage locations.

Altering an existing storage location has the following effects:

Alter use from:	To store only:	Has this effect:
Temp and data files (or data only)	Temp files	Data content is eventually merged out through ATM, per its policies. You can also merge out data from the storage location manually using DO_TM_TASK . The location stores only temp files from that point forward.
Temp and data files (or temp only)	Data files	Vertica continues to run all statements that use temp files (such as queries and loads). Subsequent statements no longer use the changed storage location for temp files, and the location stores only data files from that point forward.

Altering Location Labels

You can change the label for a storage location in the following ways:

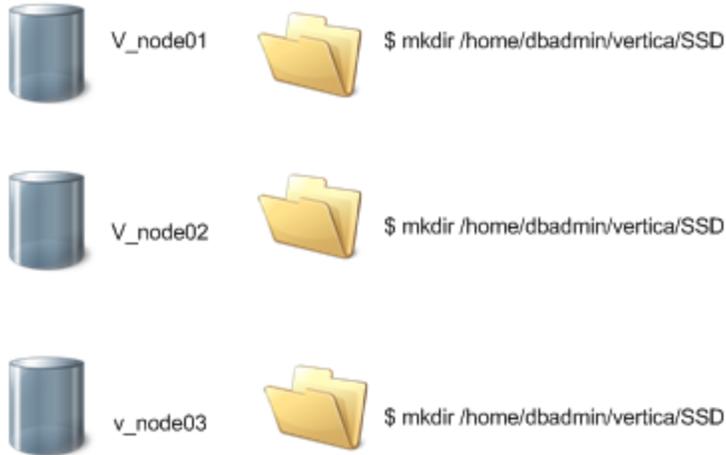
- Add a label to an unlabeled storage location.
- Change an existing label.
- Remove a label.

You can perform this operation on individual nodes (by naming them) or cluster-wide (by specifying an empty string for the node).

Important: You can label an existing storage location that already contains data. However, including the labeled location in one or more storage policies can cause the ATM to relocate existing data. This situation occurs if the ATM determines that data stored on a labeled location does not comply with a storage policy.

Adding a Location Label

To alter a location label, use the [ALTER_LOCATION_LABEL](#) function. The following figure illustrates an unlabeled location that exists on three nodes:



This example shows how to add a location label, 'SSD', to the existing storage locations on all nodes (' '):

```
=> SELECT ALTER_LOCATION_LABEL('/home/dbadmin/vertica/SSD', '', 'SSD');
```

Removing a Location Label

You cannot remove a location label if *both* of the following conditions exist:

- The name being removed is used in a storage policy
- The location from which you are removing the label is the last available storage for its associated objects

The following example removes the SSD label for the storage location on all nodes by specifying empty strings (' ') for both *node* and *location_label* parameters:

```
VMART=> select alter_location_label('/home/dbadmin/SSD/tables','', '');
          alter_location_label
-----
/home/dbadmin/SSD/tables label changed.
(1 row)
```

Effects of Altering a Location Label

Altering a location label has the following effects:

Alter label:	To:	Has this effect:
No name	New label	Lets you use the labeled storage within a storage policy. See note above regarding data storage being moved to other locations if you add a label to a storage location with existing data.
Existing name	New name	You can use the new label in a storage policy. If the existing name is used in a storage policy, you cannot change the label.
Existing name	No name	You cannot use an unlabeled storage in a storage policy. If the existing name is used in a storage policy, you cannot remove the label.

Creating Storage Policies

You create a storage policy to associate a database object with a labeled storage location. To do so, use the [SET_OBJECT_STORAGE_POLICY](#) function. When a storage policy exists, Vertica uses the labeled location as the default storage location for the object data. Storage policies let you determine where to store your critical data. For example, suppose you create a storage location with the label SSD representing the fastest available storage on the cluster nodes. You can then create storage policies to associate tables with that labeled location. One storage policy can exist per database object.

Note: You cannot include temporary files in storage policies. Storage policies are for use only with data files on storage locations for DATA. Storage policies are not valid for USER locations.

You can create a storage policy for any database object (database, schemas, tables, and partition ranges). Each time data is loaded and updated, Vertica checks to see whether the object has a storage policy. If it does, Vertica automatically uses the labeled storage location. If no storage policy exists for an object, or its parent entities, data storage processing continues using standard storage algorithms on available storage locations. If all storage locations are labeled, Vertica uses one of them.

Creating one or more storage policies does not require that policies exist for all database objects. A site can support objects with or without storage policies. You can add storage policies for a discrete set of priority objects, while letting other objects exist without a policy, so they use available storage.

Creating Policies Based on Storage Performance

You can measure the performance of any disk storage location (see [Measuring Storage Performance](#)). Then, using the performance measurements, set the storage location performance. Vertica uses the performance measurements you set to rank its storage locations and, through ranking, to determine which key projection columns to store on higher performing locations, as described in [Setting Storage Performance](#).

If you have already set the performance of your site's storage locations, and decide to use storage policies, any storage location with an associated policy has a higher priority than the storage ranking setting.

Storage Levels and Priorities

Vertica assigns storage levels to database objects. The database is the highest storage level (because nothing exists above the database level), and partition `min_` and `max_` key ranges are considered the lowest level objects. In addition to storage levels, storage priorities exist. The lower the storage level of an object, the higher its storage priority.

Consider this example of database objects, listed in storage level order, with the highest level, Sales database, first:

Object	Storage Level	Storage Policy	Storage Priority	Labeled Location
Sales (database)	Highest	YES	Lower	STANDARD
Region (schema)	Medium	NO	Medium	N/A
Income (table)	Lower	YES	Higher/highest	FAST
Month (partitions)	Lowest	NO	Highest	N/A

Storage policies exist for the database and table objects, with default storage on the locations STANDARD and FAST, respectively.

In this case, when TM operations occur, such as moveout and mergeout, table data has the highest priority. The TM moves data from WOS to ROS to the FAST labeled location.

Any schema data changes are prioritized after table data. Because the Region schema has no storage policy, Vertica searches up the storage levels for a policy. In this case, that is the Sales database itself. If a database storage policy is in effect, Region schema data is moved from

WOS to ROS to the STANDARD storage location, using its parent object's default storage location.

If the Sales database object had no storage policy, the TM operations would use existing storage locations and mechanisms.

Using the SET_OBJECT_STORAGE_POLICY Function

To set a storage policy, use the [SET_OBJECT_STORAGE_POLICY](#) function.

This example shows how to set a storage policy for the table test to use the storage labeled SSD as its default location:

```
=> select set_object_storage_policy('test','ssd', true);
set_object_storage_policy
-----
Object storage policy set.
Task: moving storages
(Table: public.test) (Projection: public.test_b0)
(Table: public.test) (Projection: public.test_b1)

(1 row)
```

You can query existing storage policies, listed in the `location_label` column of the `v_monitor.storage_containers` system table:

```
VMART=> select node_name, projection_name, storage_type, location_label from v_monitor.storage_containers;
 node_name      | projection_name | storage_type | location_label
-----|-----|-----|-----
v_vmart_node0001 | states_p       | ROS         |
v_vmart_node0001 | states_p       | ROS         |
v_vmart_node0001 | t1_b1          | ROS         |
v_vmart_node0001 | newstates_b0   | ROS         | LEVEL3
v_vmart_node0001 | newstates_b0   | ROS         | LEVEL3
v_vmart_node0001 | newstates_b1   | ROS         | LEVEL3
v_vmart_node0001 | newstates_b1   | ROS         | LEVEL3
v_vmart_node0001 | newstates_b1   | ROS         | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS         | LEVEL3
v_vmart_node0001 | states_b0      | ROS         | SSD
v_vmart_node0001 | states_b0      | ROS         | SSD
v_vmart_node0001 | states_b1      | ROS         | SSD
v_vmart_node0001 | states_b1      | ROS         | SSD
v_vmart_node0001 | states_b1      | ROS         | SSD
.
```

You can use storage policies to move older data to less-expensive storage locations while keeping it available for queries. See [Creating Storage Policies for Low-Priority Data](#).

Effects of Creating Storage Policies

Creating storage policies has the following effects:

Create policy for...	Storage effect:
Database	The highest storage level and the lowest storage priority. This is the default policy when no lower-level or higher priority policies exist. At storage time, Vertica uses the database policy for all objects without storage policies.
Schema	The mid-level storage, also with a medium priority, compared to lower storage level objects. If a table's schema has no policy, the TM searches the next higher level, the database, using that policy, if it exists. If it does not, the TM uses existing storage mechanisms.
Table	A lower storage level than a schema, with the highest storage priority, if no policy exists for the table's partition key ranges. If a table has no storage policy, Vertica checks the next higher storage level (the schema) for a policy and uses that. If the schema has no policy, it checks the next higher level, the database, and uses that. If no database policy exists, the TM uses existing storage mechanisms.
Partition min_key and max_key ranges	The lowest level policy that can be in effect. During storage processing, partition key ranges with a policy have the highest priority. If no policy exists, the parent table is checked, and so on as described for the other database objects.

Creating Storage Policies for Low-Priority Data

If some of your data is in a partitioned table, you can move less-queried partitions to less-expensive storage such as HDFS. The data is still accessible in queries, just at a slower speed. In this scenario, the faster storage is often referred to as "hot storage," and the slower storage is referred to as "cold storage."

Suppose you have a table named `messages` (containing social-media messages) that is partitioned by the year and month of the message's timestamp. You can list the partitions in the table by querying the `PARTITIONS` system table.

```
=> SELECT partition_key, projection_name, node_name, location_label FROM partitions
      ORDER BY partition_key;
partition_key | projection_name | node_name      | location_label
-----+-----+-----+-----
201309       | messages_b1    | v_vmart_node0001 |
201309       | messages_b0    | v_vmart_node0003 |
201309       | messages_b1    | v_vmart_node0002 |
201309       | messages_b1    | v_vmart_node0003 |
201309       | messages_b0    | v_vmart_node0001 |
201309       | messages_b0    | v_vmart_node0002 |
201310       | messages_b0    | v_vmart_node0002 |
201310       | messages_b1    | v_vmart_node0003 |
201310       | messages_b0    | v_vmart_node0001 |
. . .
201405       | messages_b0    | v_vmart_node0002 |
201405       | messages_b1    | v_vmart_node0003 |
201405       | messages_b1    | v_vmart_node0001 |
201405       | messages_b0    | v_vmart_node0001 |
(54 rows)
```

Next, suppose you find that most queries on this table access only the latest month or two of data. You might decide to move the older data to cold storage in an HDFS-based storage location. After you move the data, it is still available for queries, but with lower query performance.

To move partitions to the HDFS storage location, supply the lowest and highest partition key values to be moved in the `SET_OBJECT_STORAGE_POLICY` function call. The following example shows how to move data between two dates. In this example:

- The partition key value 201309 represents September 2013.
- The partition key value 201403 represents March 2014.
- The name, `coldstorage`, is the label of the HDFS-based storage location.
- The final argument, which is optional, is `true`, meaning that the function does not return until the move is complete. By default the function returns immediately and the data is moved when the Tuple Mover next runs. When data is old, however, the Tuple Mover runs less frequently, which would delay recovering the original storage space.

```
=> SELECT SET_OBJECT_STORAGE_POLICY('messages','coldstorage', '201309', '201403', 'true');
```

The partitions within the specified range are moved to the HDFS storage location labeled `coldstorage` the next time the Tuple Mover runs. This location name now displays in the `PARTITIONS` system table's `location_label` column.

```
=> SELECT partition_key, projection_name, node_name, location_label
      FROM partitions ORDER BY partition_key;
partition_key | projection_name | node_name | location_label
-----+-----+-----+-----
201309       | messages_b0    | v_vmart_node0003 | coldstorage
201309       | messages_b1    | v_vmart_node0001 | coldstorage
201309       | messages_b1    | v_vmart_node0002 | coldstorage
201309       | messages_b0    | v_vmart_node0001 | coldstorage
. . .
201403       | messages_b0    | v_vmart_node0002 | coldstorage
201404       | messages_b0    | v_vmart_node0001 |
201404       | messages_b0    | v_vmart_node0002 |
201404       | messages_b1    | v_vmart_node0001 |
201404       | messages_b1    | v_vmart_node0002 |
201404       | messages_b0    | v_vmart_node0003 |
201404       | messages_b1    | v_vmart_node0003 |
201405       | messages_b0    | v_vmart_node0001 |
201405       | messages_b1    | v_vmart_node0002 |
201405       | messages_b0    | v_vmart_node0002 |
201405       | messages_b0    | v_vmart_node0003 |
201405       | messages_b1    | v_vmart_node0001 |
201405       | messages_b1    | v_vmart_node0003 |
(54 rows)
```

After your initial data move, you can move additional data to the HDFS storage location periodically. You can move individual partitions or a range of partitions from the "hot" storage to the "cold" storage location using the same method:

```
=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'coldstorage', '201404', '201404', 'true');

=> SELECT projection_name, node_name, location_label
      FROM PARTITIONS WHERE PARTITION_KEY = '201404';
projection_name | node_name | location_label
-----+-----+-----
messages_b0     | v_vmart_node0002 | coldstorage
messages_b0     | v_vmart_node0003 | coldstorage
messages_b1     | v_vmart_node0003 | coldstorage
messages_b0     | v_vmart_node0001 | coldstorage
messages_b1     | v_vmart_node0002 | coldstorage
messages_b1     | v_vmart_node0001 | coldstorage
(6 rows)
```

Moving Partitions to a Table Stored on HDFS

Another method of moving partitions from hot storage to cold storage is to move the partitions' data to a separate table in the other storage location. This method breaks the data into two tables, one containing hot data and the other containing cold data. Use this method if you want to prevent queries from inadvertently accessing data stored in cold storage. To query the older data, you must explicitly query the cold table.

To move partitions:

1. Create a new table whose schema matches that of the existing partitioned table.
2. Set the storage policy of the new table to use the HDFS-based storage location.
3. Use the [MOVE_PARTITIONS_TO_TABLE](#) function to move a range of partitions from the hot table to the cold table. The partitions migrate when the Tuple Mover next runs.

The following example demonstrates these steps. You first create a table named `cold_messages`. You then assign it the HDFS-based storage location named `coldstorage`, and, finally, move a range of partitions.

```
=> CREATE TABLE cold_messages LIKE messages INCLUDING PROJECTIONS;  
=> SELECT SET_OBJECT_STORAGE_POLICY('cold_messages', 'coldstorage');  
=> SELECT MOVE_PARTITIONS_TO_TABLE('messages', '201309', '201403', 'cold_messages');
```

Moving Data Storage Locations

You can use the [SET_OBJECT_STORAGE_POLICY](#) function to move data storage from an existing location (whether labeled or not) to another labeled location. You can use this function to accomplish two tasks:

1. Create or update a storage policy:
 - Create new storage policy for the object
 - Update an existing policy to the target labeled location.
2. Move all existing data for the specified objects to the target storage location.

Vertica moves the existing data the next time the Tuple Mover runs. Alternatively, you can enforce the data move to occur in the current transaction using the function's `enforce_storage_move` parameter. Consider setting this parameter to `true` if the Tuple Mover will not run on this data for a while, such as if the data is old.

Before actually moving the object to the target storage location, Vertica calculates the required storage and checks available space at the target. If there is insufficient free space, the function generates an error and stops execution. It does not attempt to find sufficient storage at another location.

Important: You should check available space on the new target location before starting a data move. However, be aware that checking does not guarantee that this space remains available during move execution. Checking target space does prevent attempts to move

any data, if insufficient space is available.

Moving Data Storage While Setting a Storage Policy

The following example shows how to use [SET_OBJECT_STORAGE_POLICY](#) to set a storage policy for the table object `states`. Then, you can move the table's existing stored data to the labeled location, `SSD`. You force the move to occur during the function transaction by specifying the last parameter as `true`:

```
=> select set_object_storage_policy('states', 'SSD', 'true');
       set_object_storage_policy
```

```
-----
Object storage policy set.
Task: moving storages
(Table: public.states) (Projection: public.states_p1)
(Table: public.states) (Projection: public.states_p2)
(Table: public.states) (Projection: public.states_p3)
(1 row)
```

Note: Moving an object's current storage to a new target is a cluster-wide operation, so a failure on any node results in a warning message. The function then attempts to continue executing on other cluster nodes.

You can view the storage policies that are in effect:

```
=> select * from storage_policies;
 schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
 public      | states      | Table          | SSD
(1 row)
```

Effects of Moving a Storage Location

Moving an object from one labeled storage location to another has the following effects:

Object type:	Effect:
Schema or table	If data storage exists, moves data from source to target destination. Source data can reside on a labeled or unlabeled storage location, but will be moved to specified labeled location. Cluster nodes unavailable when existing data is copied are updated by the TM when they rejoin the cluster. Alternately, you can enforce a data

Object type:	Effect:
	<p>move as part of the function transaction, by specifying the last parameter as <code>true</code>.</p> <p>If a storage policy was in effect, the default storage location changes from the source to target location. This change affects all future TM operations, such as moveout and mergeout activities.</p>
Table with specified partition min-keys and max_keys	Sets a policy or moves existing data only for the <code>key_min</code> and <code>key_max</code> ranges. Separate partition key ranges can have different storage policies from other ranges or the parent table.

Clearing Storage Policies

You can clear a storage policy by object name after you have defined storage policies. To see existing policies, query the `storage_policies` system table, described in [Viewing Storage Locations and Policies](#).

To clear a storage policy, use the `CLEAR_OBJECT_STORAGE_POLICY` function, specifying the object name associated with the labeled location:

```
=> select clear_object_storage_policy('lineorder');      clear_object_storage_policy
-----
Default storage policy cleared.
(1 row)
```

Existing data is moved to the parent storage policy's location, or the default storage location if there is no parent policy. Typically, the move occurs the next time the Tuple Mover runs. Alternatively, you can force the data move to occur in the current transaction by setting the optional `enforce_storage_move` parameter to `true`.

You can also use the `ENFORCE_OBJECT_STORAGE_POLICY` meta-function to trigger the move for all storage locations at once. Using this function equates to setting `enforce_storage_move`.

Speeding Up Data Migration

After you clear the storage policy, the Tuple Mover eventually migrates the object's data from the storage location to the database's default storage location. The TM moves the data when

it performs a move-storage operation. This operation runs infrequently at low priority. Therefore, it might be some time before the data migrates out of the storage location.

You can speed up the data-migration process by:

1. Calling the [RETIRE_LOCATION](#) function to retire the storage location on each host that defines it.
2. Calling the [MOVE_RETIRED_LOCATION_DATA](#) function to move the location's data to the database's default storage location.
3. Calling the [RESTORE_LOCATION](#) function to restore the location on each host that defines it. You must perform this step because you cannot drop retired storage locations.

The following example demonstrates clearing the object storage policy of a table stored on HDFS, then performing the steps to move the data off of the location. (See [Using HDFS Storage Locations](#) for additional background.)

```
=> SELECT * FROM storage_policies;
  schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public      | test       | Table         | hdfs2
(1 row)

=> SELECT clear_object_storage_policy('test');
  clear_object_storage_policy
-----
Object storage policy cleared.
(1 row)

=> SELECT retire_location('webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0001',
  'v_vmart_node0001');
  retire_location
-----
webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0001 retired.
(1 row)

=> SELECT retire_location('webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0002',
  'v_vmart_node0002');
  retire_location
-----
webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0002 retired.
(1 row)

=> SELECT retire_location('webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0003',
  'v_vmart_node0003');
  retire_location
-----
webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0003 retired.
(1 row)

=> SELECT node_name, projection_name, location_label, total_row_count FROM
  V_MONITOR.STORAGE_CONTAINERS WHERE projection_name ILIKE 'test%';
  node_name      | projection_name | location_label | total_row_count
-----+-----+-----+-----
```

```
v_vmart_node0001 | test_b1      | hdfs2      |          333631
v_vmart_node0001 | test_b0      | hdfs2      |          332233
v_vmart_node0002 | test_b1      | hdfs2      |          332233
v_vmart_node0002 | test_b0      | hdfs2      |          334136
v_vmart_node0003 | test_b1      | hdfs2      |          334136
v_vmart_node0003 | test_b0      | hdfs2      |          333631
(6 rows)
```

```
=> SELECT move_retired_location_data();
       move_retired_location_data
```

```
-----
Move data off retired storage locations done
```

```
(1 row)
```

```
=> SELECT node_name, projection_name, location_label, total_row_count
       FROM V_MONITOR.STORAGE_CONTAINERS WHERE projection_name ILIKE 'test%';
       node_name      | projection_name | location_label | total_row_count
```

```
-----+-----+-----+-----
v_vmart_node0001 | test_b0      |          |          332233
v_vmart_node0001 | test_b1      |          |          333631
v_vmart_node0002 | test_b0      |          |          334136
v_vmart_node0002 | test_b1      |          |          332233
v_vmart_node0003 | test_b0      |          |          333631
v_vmart_node0003 | test_b1      |          |          334136
(6 rows)
```

```
=> SELECT restore_location('webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0001',
       'v_vmart_node0001');
```

```
restore_location
```

```
-----
webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0001 restored.
```

```
(1 row)
```

```
=> SELECT restore_location('webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0002',
       'v_vmart_node0002');
```

```
restore_location
```

```
-----
webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0002 restored.
```

```
(1 row)
```

```
=> SELECT restore_location('webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0003',
       'v_vmart_node0003');
```

```
restore_location
```

```
-----
webhdfs://hadoop:50070/user/dbadmin/v_vmart_node0003 restored.
```

```
(1 row)
```

Effects on Same-Name Storage Policies

The effects of clearing a storage policy depend on which policy you clear.

For example, consider the following storage configuration. The table `lineorder` has a storage policy for default storage to the location label F2. The table's partition ranges, also `lineorder` objects, have storage policies for other default storage locations:

```
=> select * from v_monitor.storage_policies;
schema_name      | object_name      | policy_details  | location_label
-----+-----+-----+-----
public           | public           | Schema         | F4
public           | lineorder        | Table          | F2
public           | lineorder        | Partition [0, 0] | F1
public           | lineorder        | Partition [1, 1] | F2
public           | lineorder        | Partition [2, 2] | F4
public           | lineorder        | Partition [3, 3] | M1
public           | lineorder        | Partition [4, 4] | M3
(7 rows)
```

For this example, clearing the storage policy for an object named `lineorder`, removes the policy for the table, while retaining storage policies for its partitions, which have their own policies.

The function determines which `lineorder` object policy to clear because no partition range values are specified in the function call:

```
=> select clear_object_storage_policy('lineorder');
clear_object_storage_policy
-----
Default storage policy cleared.
(1 row)
release=> select * from v_monitor.storage_policies;
schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public      | public      | Schema         | F4
public      | lineorder   | Partition [0, 0] | F1
public      | lineorder   | Partition [1, 1] | F2
public      | lineorder   | Partition [2, 2] | F4
public      | lineorder   | Partition [3, 3] | M1
public      | lineorder   | Partition [4, 4] | M3
(6 rows)
```

Further, using a partition key range with the `lineorder` object name clears the storage policy for only the specified partition range(s). The storage policy for the parent table objects, and other partition ranges persist:

```
=> select clear_object_storage_policy ('lineorder','0','3');
clear_object_storage_policy
-----
Default storage policy cleared.
(1 row)
release=> select * from storage_policies;
schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public      | public      | Schema         | F4
public      | lineorder   | Table          | F2
public      | lineorder   | Partition [4, 4] | M3
```

(2 rows)

Measuring Storage Performance

Vertica lets you measure disk I/O performance on any storage location at your site. You can use the returned measurements to set performance, which automatically provides rank.

Depending on your storage needs, you can also use performance to determine the storage locations needed for critical data as part of your site's storage policies. Storage performance measurements apply only to data storage locations, not temporary storage locations.

Measuring storage location performance calculates the time it takes to read and write 1 MB of data from the disk, which equates to:

$$\text{IO time} = \text{time to read/write 1MB} + \text{time to seek} = 1/\text{throughput} + 1/\text{Latency}$$

- Throughput is the average throughput of sequential reads/writes (expressed in megabytes per second).
- Latency is for random reads only in seeks (units in seeks per second).

Thus, the I/O time of a faster storage location is less than that of a slower storage location.

Note: Measuring storage location performance requires extensive disk I/O, which is a resource-intensive operation. Consider starting this operation when fewer other operations are running.

Vertica gives you two ways to measure storage location performance, depending on whether the database is running. You can either:

- Measure performance on a running database.
- Measure performance before a cluster is set up.

Both methods return the throughput and latency for the storage location. Record or capture the throughput and latency information so you can use it to set the location performance (see [Setting Storage Performance](#)).

Measuring Performance on a Running Vertica Database

Use the [MEASURE_LOCATION_PERFORMANCE\(\)](#) function to measure performance for a storage location when the database is running. This function has the following requirements:

- The storage path must already exist in the database.
- You need RAM*2 free space available in a storage location to measure its performance. For example, if you have 16 GB RAM, you need 32 GB of available disk space. If you do not have enough disk space, the function returns an error.

Use the system table [DISK_STORAGE](#) to obtain information about disk storage on each database node.

The following example shows how to measure the performance of a storage location on v_vmartdb_node0004:

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/', 'v_vmartdb_node0004');  
WARNING: measure_location_performance can take a long time. Please check logs for progress  
measure_location_performance  
-----  
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

Measuring Performance Before a Cluster Is Set Up

You can measure disk performance before setting up a cluster. This approach is useful when you want to verify that the disk is functioning within normal parameters. To perform this measurement, you must already have Vertica installed.

To measure disk performance, use the following command:

```
opt/vertica/bin/vertica -m <path to disk mount>
```

For example:

```
opt/vertica/bin/vertica -m /secondVerticaStorageLocation/node0004_data
```

Setting Storage Performance

You can use the measurements returned from the [MEASURE_LOCATION_PERFORMANCE](#) function as input values to the [SET_LOCATION_PERFORMANCE\(\)](#) function.

Note: You must set the throughput and latency parameters of this function to 1 or more.

The following example shows how to set the performance of a storage location on `v_`
`vmartdb_node0004`, using values for this location returned from the [MEASURE_LOCATION_](#)
[PERFORMANCE](#) function. Set the throughput to 122 MB/second and the latency to 140
seeks/second. [MEASURE_LOCATION_PERFORMANCE](#)

```
=> SELECT SET_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/', 'node2', '122', '140');
```

Sort Order Ranking by Location Performance Settings

After you set performance-data parameters, , Vertica automatically uses performance data to rank storage locations whenever it stores projection columns.

Vertica stores columns included in the projection sort order on the fastest available storage locations. Columns not included in the projection sort order are stored on slower disks. Columns for each projection are ranked as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next-to-last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given 1000 + # of sort columns.
- The remaining columns are given numbers from 1000–1, starting with 1000 and decrementing by one per column.

Vertica then stores columns on disk from the highest ranking to the lowest ranking. It places highest-ranking columns on the fastest disks and the lowest-ranking columns on the slowest disks.

Using Location Performance Settings with Storage Policies

You initially measure location performance and set it in the Vertica database. Then, you can use the performance results to determine the fastest storage to use in your storage policies.

- Set the locations with the highest performance as the default locations for critical data.
- Use slower locations as default locations for older, or less-important data. Such slower locations may not require policies at all, if you do not want to specify default locations.

Vertica determines data storage as follows, depending on whether a storage policy exists:

Storage Policy	Label	Number of Locations	Vertica Action
No	No	Multiple	Uses ranking (as described), choosing a location from all locations that exist.
Yes	Yes	Single	Uses that storage location exclusively.
Yes	Yes	Multiple	Ranks storage (as described) among all same-name labeled locations.

Dropping Storage Locations

To drop a storage location, use the [DROP_LOCATION\(\)](#) function. The following example shows how to drop a storage location on `v_vmartdb_node0002` that was used to store temp files:

```
=> SELECT DROP_LOCATION('/secondVerticaStorageLocation/' , 'v_vmartdb_node0002');
```

When you drop a storage location, the operation cascades to associated objects including any granted privileges to the storage.

Caution: Dropping a storage location is a permanent operation and cannot be undone. Subsequent queries on storage used for external table access fail with a COPY COMMAND FAILED message.

Because dropping a storage location cannot be undone, Micro Focus recommends that you first retire a storage location (see [Retiring Storage Locations](#)). Retiring a storage location before dropping it lets you verify that there will be no adverse effects on any data access. Additionally, you can restore a retired storage location (see [Restoring Retired Storage Locations](#)).

Altering Storage Locations Before Dropping Them

You can drop only storage locations containing temp files. Thus, you must alter a storage location to the TEMP usage type before you can drop it. However, if data files still exist on the storage, Vertica prevents you from dropping the storage location. Deleting data files does not clear the storage location and can result in database corruption. To handle a storage area containing data files so that you can drop it, use one of these options:

- Manually merge out the data files.
- Wait for the ATM to mergeout the data files automatically.
- Retire the location, and force the changes to take effect immediately.
- Manually [drop partitions](#)

Dropping USER Storage Locations

Storage locations that you create with the USER usage type can contain only data files, not temp files. However, you can drop a USER location, regardless of any remaining data files. This behavior differs from that of a storage location not designated for USER access.

Checking Location Properties

You can check the properties of a storage location, such as whether it is a USER location or is being used only for TEMP files, in the [STORAGE_LOCATIONS](#) system table. You can also use this table to verify that a location has been retired.

Retiring Storage Locations

To retire a storage location, use the [RETIRE_LOCATION\(\)](#) function.

The following example shows how to retire a storage location on v_vmartdb_node0004:

```
=> SELECT RETIRE_LOCATION('/secondStorageLocation/' , 'v_vmartdb_node0004');
```

To retire a storage location on all nodes, use an empty string (") for the second parameter.

Retiring a location prevents Vertica from storing data or temp files to it, but does not remove the actual location. Any data previously stored on the retired location is eventually merged out by the Automatic Tuple Mover (ATM) per its policies.

If you plan to drop the storage location after retiring it, you can expedite this by setting the optional `enforce_storage_move` parameter to `true`. This setting moves the data out of the storage location instead of waiting for the Tuple Mover, allowing you to drop the location immediately.

You can also use the [ENFORCE_OBJECT_STORAGE_POLICY](#) meta-function to trigger the move for all storage locations at once, and then drop the locations. This approach equates to setting `enforce_storage_move`.

The following example shows how to retire a storage location on all nodes and prepares it for immediate drop:

```
=> SELECT RETIRE_LOCATION('/secondStorageLocation/' , '', true);
```

Note: If the location used in a storage policy is the last available storage for its associated objects, you cannot retire it *unless* you set `enforce_storage_move` to `true`.

Data and temp files can be stored in one, or multiple separate, storage locations.

For further information on dropping a location after retiring it, see [Dropping Storage Locations](#).

Restoring Retired Storage Locations

You can restore a previously retired storage location that continues to be used in queries. After the location is restored, Vertica re-ranks the storage location and uses the restored location to process queries as determined by its rank.

Use the [RESTORE_LOCATION\(\)](#) function to restore a retired storage location.

The following example shows how to restore a retired storage location on `v_vmartdb_node0004`:

```
=> SELECT RESTORE_LOCATION('/secondVerticaStorageLocation/' , 'v_vmartdb_node0004');
```

To restore a storage location on all nodes, use an empty string (") for the second parameter. The following example demonstrates creating, retiring, and restoring a location on all nodes:

```
=> create location '/tmp/ab1' all nodes usage 'TEMP';  
CREATE LOCATION  
  
=> select retire_location('/tmp/ab1', '');
```

```

retire_location
-----
/tmp/ab1 retired.
      (1 row)

=> select * from storage_locations where location_path ilike '/tmp/ab1';
location_id | node_name | location_path | location_usage | is_retired | location_
label | rank | throughput | latency
-----+-----+-----+-----+-----+-----
--+-----+-----+-----+-----+-----+-----
45035996273736724 | v_vmart_node0001 | /tmp/ab1 | TEMP | t |
| 0 | 0 | 0
45035996273736726 | v_vmart_node0002 | /tmp/ab1 | TEMP | t |
| 0 | 0 | 0
45035996273736728 | v_vmart_node0003 | /tmp/ab1 | TEMP | t |
| 0 | 0 | 0
45035996273736730 | v_vmart_node0004 | /tmp/ab1 | TEMP | t |
| 0 | 0 | 0
      (4 rows)

=> select restore_location('/tmp/ab1', '');
restore_location
-----
/tmp/ab1 restored.
      (1 row)

=> select * from storage_locations where location_path ilike '/tmp/ab1';
location_id | node_name | location_path | location_usage | is_retired | location_
label | rank | throughput | latency
-----+-----+-----+-----+-----+-----
--+-----+-----+-----+-----+-----+-----
45035996273736724 | v_vmart_node0001 | /tmp/ab1 | TEMP | f |
| 0 | 0 | 0
45035996273736726 | v_vmart_node0002 | /tmp/ab1 | TEMP | f |
| 0 | 0 | 0
45035996273736728 | v_vmart_node0003 | /tmp/ab1 | TEMP | f |
| 0 | 0 | 0
45035996273736730 | v_vmart_node0004 | /tmp/ab1 | TEMP | f |
| 0 | 0 | 0
      (4 rows)

```

The `RESTORE_LOCATION()` function restores the location only on the nodes where the location exists and is retired. The function does not propagate the storage location to nodes where that location did not previously exist.

Restoring on all nodes fails if the location has been dropped on any of them. If you have dropped the location on some nodes, you have two options:

- If you no longer want to use the dropped node, restore the location individually on each of the other nodes.
- Alternatively, you can re-create the location on the node where you dropped it. To do so, use `CREATE LOCATION()` or `ADD_LOCATION()`. After you have re-created the location, you can then restore it on all nodes.

The following example demonstrates the failure if you try to restore on nodes where you have dropped the location:

```
=> select retire_location('/tmp/ab1', '');
retire_location
-----
/tmp/ab1 retired.
      (1 row)

=> select drop_location('/tmp/ab1', 'v_vmart_node0002');
drop_location
-----
/tmp/ab1 dropped.
      (1 row)

=> select * from storage_locations where location_path ilike '/tmp/ab1';
location_id      | node_name          | location_path | location_usage | is_retired | location_
label | rank | throughput | latency
-----+-----+-----+-----+-----+-----
---+---+---+---+---+---
45035996273736724 | v_vmart_node0001  | /tmp/ab1     | TEMP          | t          |
| 0 | 0 | 0
45035996273736728 | v_vmart_node0003  | /tmp/ab1     | TEMP          | t          |
| 0 | 0 | 0
45035996273736730 | v_vmart_node0004  | /tmp/ab1     | TEMP          | t          |
| 0 | 0 | 0
      (3 rows)

=> select restore_location('/tmp/ab1', '');
ERROR 2081: [/tmp/ab1] is not a valid storage location on node v_vmart_node0002
```

Analyzing Workloads

If queries perform suboptimally, you can get tuning recommendations for them, as well as for hints about optimizing database objects, by using the Workload Analyzer (WLA). WLA is a Vertica utility that analyzes system information in Vertica [system tables](#).

WLA identifies the root causes of poor query performance through intelligent monitoring of query execution, workload history, resources, and configurations. It then returns a set of tuning recommendations based on statistics, system and data collector events, and database/table/projection design. You can use these recommendations to tune query performance, quickly and easily.

You can run WLA in two ways:

- Call the Vertica function [ANALYZE_WORKLOAD](#).
- Use the [Management Console interface](#).

See [Understanding WLA Triggering Conditions](#) for the most common triggering conditions and recommendations.

Getting Tuning Recommendations

Call the function [ANALYZE_WORKLOAD](#) to get tuning recommendations for queries and database objects. The function arguments specify what events to analyze and when.

Setting Scope and Time Span

`ANALYZE_WORKLOAD`'s `scope` argument determines what to analyze:

This argument...	Returns WLA recommendations for...
' ' (empty string)	All database objects
Table name	A specific table
Schema name	All objects in the specified schema

The optional `since-time` argument specifies to return values from all in `-scope` events starting from `since-time` and continuing to the current system status. If you omit `since-time`, `ANALYZE_WORKLOAD` returns recommendations for events since the last recorded time

that you called the function. You must explicitly cast the since-time string value to either `TIMESTAMP` or `TIMESTAMPTZ`.

The following examples show four ways to express the since-time argument with different formats. All queries return the same result for workloads on table `t1` since October 4, 2012:

```
=> SELECT ANALYZE_WORKLOAD('t1', TIMESTAMP '2012-10-04 11:18:15');  
=> SELECT ANALYZE_WORKLOAD('t1', '2012-10-04 11:18:15':TIMESTAMPTZ);  
=> SELECT ANALYZE_WORKLOAD('t1', 'October 4, 2012':TIMESTAMP);  
=> SELECT ANALYZE_WORKLOAD('t1', '10-04-12':TIMESTAMPTZ);
```

Saving Function Results

Instead of analyzing events since a specific time, you can save results from `ANALYZE_WORKLOAD`, by setting the function's second argument to `true`. The default is `false`, and no results are saved. After saving function results, subsequent calls to `ANALYZE_WORKLOAD` analyze only events since you last saved returned data, and ignore all previous events.

For example, the following statement returns recommendations for all database objects in all schemas and records this analysis invocation.

```
=> SELECT ANALYZE_WORKLOAD('', true);
```

The next invocation of `ANALYZE_WORKLOAD` analyzes events from this point forward.

```
-[ RECORD 1 ]-----+-----  
observation_count | 1  
first_observation_time |  
last_observation_time |  
tuning_parameter | public.locations  
tuning_description | run database designer on table public.locations  
tuning_command |  
tuning_cost | HIGH  
-[ RECORD 2 ]-----+-----  
observation_count | 1  
first_observation_time |  
last_observation_time |  
tuning_parameter | public.new_locations  
tuning_description | run database designer on table public.new_locations  
tuning_command |  
tuning_cost | HIGH  
-[ RECORD 3 ]-----+-----  
observation_count | 1  
first_observation_time |  
last_observation_time |  
tuning_parameter | release  
tuning_description | set password for user 'release'  
tuning_command | alter user release identified by 'new_password'  
tuning_cost | LOW
```

Observation Count and Time

The `observation_count` column returns an integer that represents the total number of events Workload Analyzer (WLA) observed for this tuning recommendation. In each case above, WLA is making its first recommendation. Null results in `observation_time` only mean that the recommendations are from the current system status instead of from a prior event.

Tuning Targets

The `tuning_parameter` column returns the object on which WLA recommends that you apply the tuning action. The parameter of `release` in the example above notifies the DBA to set a password for user release.

Tuning Recommendations and Costs

WLA's output returns a brief description of tasks you should consider in the `tuning_description` column, along with a SQL command you can run, where appropriate, in the `tuning_command` column. In records 1 and 2 above, WLA recommends that you run the Database Designer on two tables, and in record 3 recommends setting a user's password. Record 3 also provides the `ALTER USER` command to run because the tuning action is a SQL command.

Output in the `tuning_cost` column indicates the cost of running the recommended tuning command:

- **LOW:** Running the tuning command has minimal impact on resources. You can perform the tuning operation at any time, like changing the user's password in Record 3 above.
- **MEDIUM:** Running the tuning command has moderate impact on resources.
- **HIGH:** Running the tuning command has maximum impact on resources. Depending on the size of your database or table, consider running high-cost operations during off-peak load times.

Examples

The following statement tells WLA to analyze all events for the `locations` table:

```
=> SELECT ANALYZE_WORKLOAD('locations');
```

WLA returns with a recommendation that you run the Database Designer on the table, an operation that, depending on the size of `locations`, might incur a high cost:

```
-[ RECORD 1 ]-----+-----  
observation_count      | 1  
first_observation_time |  
last_observation_time  |  
tuning_parameter      | public.locations  
tuning_description     | run database designer on table public.locations  
tuning_command         |  
tuning_cost           | HIGH
```

The following statement analyzes workloads on all tables in the VMart example database since one week before today:

```
=> SELECT ANALYZE_WORKLOAD('', NOW() - INTERVAL '1 week');
```

WLA returns with the following results:

```
-[ RECORD 1 ]-----+-----  
observation_count      | 4  
first_observation_time | 2012-02-17 13:57:17.799003-04  
last_observation_time  | 2011-04-22 12:05:26.856456-04  
tuning_parameter      | store.store_orders_fact.date_ordered  
tuning_description     | analyze statistics on table column store.store_orders_fact.date_ordered  
tuning_command         | select analyze_statistics('store.store_orders_fact.date_ordered');  
tuning_cost           | MEDIUM  
-[ RECORD 2 ]-----+-----  
...  
-[ RECORD 14 ]-----+-----  
observation_count      | 2  
first_observation_time | 2012-02-19 17:52:03.022644-04  
last_observation_time  | 2012-02-19 17:52:03.02301-04  
tuning_parameter      | SELECT x FROM t WHERE x > (SELECT SUM(DISTINCT x) FROM  
| t GROUP BY y) OR x < 9;  
tuning_description     | consider incremental design on query  
tuning_command         |  
tuning_cost           | HIGH
```

WLA finds two issues:

- In record 1, the `date_ordered` column in the `store.store_orders_fact` table likely has [stale statistics](#), so WLA suggests running [ANALYZE_STATISTICS](#) on that column. The function output also returns the query to run. For example:

```
=> SELECT ANALYZE_STATISTICS('store.store_orders_fact.date_ordered');
```

- In record 14, WLA identifies an under-performing query in the `tuning_parameter` column. It recommends to use the Database Designer to run an incremental design. WLA rates the potential cost as HIGH.

System Table Recommendations

You can also get tuning recommendations by querying system table [TUNING_RECOMMENDATIONS](#), which returns tuning recommendation results from the last `ANALYZE_WORKLOAD` call.

```
=> SELECT * FROM tuning_recommendations;
```

System information that WLA uses for its recommendations is held in [SQL system tables](#), so querying the `TUNING_RECOMMENDATIONS` system table does not run WLA.

See Also

[Collecting Database Statistics](#)

Understanding WLA Triggering Conditions

Workload Analyzer (WLA) monitors suspicious system activity and makes recommendations based on its observations. When you run Workload Analyzer, the utility returns the following information:

- The tuning description
- The objects on which WLA applies tuning action
- The suggested SQL command for you to run (where appropriate)

In rare circumstances, tuning recommendation WLA proposes might not resolve the underlying problem. Because you might occasionally need help understanding WLA's recommendations, the following section lists some of the most common triggering conditions, along with the recommendations to resolve the issue and a pointer to more information, when available.

Common Problems

Problem

Internal configuration parameter is not the same across nodes.

Solution

Reset configuration parameter.

```
=> ALTER DATABASE mydb SET parameter = value;
```

Problem

An unused projection meets the following conditions:

- No queries on projection for more than 30 days but, projection's anchor table has been queried more than 10 times
- Projection's anchor table is not a temp or system table
- Projection's table is not small, where the number of bytes of disk storage in use by the projection (`used_bytes`) is more than 10M

Solution

Drop the projection (*projection-name*).

```
=> DROP PROJECTION public.T1_fact_super_P1_B1;
```

Problem

User with `dbadmin/pseudosuperuser` role has empty password.

Solution

Set the password for user.

```
=> ALTER USER (user) IDENTIFIED BY ('new_password');
```

Problem

Table with too many partition count.

Solution

Alter the table's partition expression.

```
=> ALTER TABLE (schema.table) PARTITION BY (new_partition_expression) REORGANIZE;
```

Problem

LGE threshold setting is lower than the default setting.

Solution

Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem

Tuple Mover's [MoveOutInterval](#) parameter is set greater than the default setting.

Solution

Decrease the [MoveOutInterval](#) configuration parameter setting.

```
=> ALTER DATABASE mydb SET MoveOutInterval = default-value;
```

For more information, see [Monitoring Events](#) and [ACTIVE_EVENTS](#).

Problem

Tuple Mover has been disabled.

Solution

Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem

Too many ROS containers since the last mergeout operation; configuration parameters are set lower than the default.

Solution

Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem

Too many ROS containers since the last mergeout operation; the TM Mergeout service is disabled.

Solution

Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem

Average CPU usage exceeds 95% for 20 minutes.

Solution

Check system processes or change the settings for `PLANNEDCONCURRENCY` and/or `MAXCONCURRENCY` for the resource pool. For details, see [ALTER RESOURCE POOL](#) and [Built-In Pool Configuration](#).

Problem

Partitioned table data is not fully reorganized after repartitioning.

Solution

Reorganize data in partitioned table public.T1.

```
=> ALTER TABLE public.T1 REORGANIZE;
```

Problem

Table has multiple partition keys within the same ROS container.

Solution

Reorganize data in partitioned table public.T1.

```
=> ALTER TABLE public.T1 REORGANIZE;
```

Problem

Excessive swap activity; average memory usage exceeds 99% for 10 minutes.

Solution

Check system processes

Problem

A table does not have any Database Designer-designed projections.

Solution

Run database designer on table public.T1. For details, see [Incremental Design](#).

Problem

Statistics are stale (no histogram or predicate falls outside histogram).

Solution

Run [ANALYZE_STATISTICS](#) on table columns.

```
=> SELECT analyze_statistics ('public.t.a');
```

See also [Collecting Database Statistics](#).

Problem

Data distribution in segmented projection is skewed.

Solution

Re-segment projection public.t_super on high-cardinality column(s). For details, see [Designing for Segmentation](#).

Problem

GROUP BY spill event.

Solution

Consider running an [incremental design](#) on query.

Managing the Database

This section describes how to manage the Vertica database. It includes the following topics:

- [Connection Load Balancing](#)
- [Managing Nodes](#)
- [Adding Disk Space to a Node](#)
- [Tuple Mover Operations](#)
- [Managing the Tuple Mover](#)
- [Managing Workloads](#)

Connection Load Balancing

Each client connection to a host in the Vertica cluster requires a small overhead in memory and processor time. If many clients connect to a single host, this overhead can begin to affect the performance of the database. You can attempt to spread the overhead of client connections by dictating that certain clients connect to specific hosts in the cluster. However, this manual balancing becomes difficult as new clients and hosts are added to your environment.

Connection load balancing helps automatically spread the overhead caused by client connections across the cluster by having hosts redirect client connections to other hosts. By redirecting connections, the overhead from client connections is spread across the cluster without having to manually assign particular hosts to individual clients. Clients can connect to a small handful of hosts, and they are naturally redirected to other hosts in the cluster.

Native Connection Load Balancing Overview

Native connection load balancing is a feature built into the Vertica Analytics Platform server and client libraries as well as vsql. Both the server and the client need to enable load balancing for it to function. If connection load balancing is enabled, a host in the database cluster can redirect a client's attempt to it to another currently-active host in the cluster. This redirection is based on a load balancing policy. This redirection can only take place once, so a client is not bounced from one host to another.

Since native connection load balancing is incorporated into the Vertica client libraries, any client application that connects to Vertica transparently takes advantage of it simply by setting a connection parameter.

How you choose to implement connection load balancing depends on your network environment. Since native load connection balancing is easier to implement, you should use it unless your network configuration requires that clients be separated from the hosts in the Vertica database by a firewall.

For more about native connection load balancing, see [About Native Connection Load Balancing](#).

About Native Connection Load Balancing

Native connection load balancing is a feature built into the Vertica server and client libraries that helps spread the CPU and memory overhead caused by client connections across the hosts in the database. It can prevent unequal distribution of client connections among hosts in the cluster.

Native connection load balancing must be enabled by the server and client. When native connection load balancing is enabled on both, the following process takes place when the client attempts to open a connection to Vertica:

1. The client connects to a host in the database cluster, with a connection parameter indicating that it is requesting a load-balanced connection.
2. The host chooses a host from the list of currently up hosts in the cluster, according to the [current load balancing scheme](#).
3. The host tells the client which host it selected to handle the client's connection.
4. If the host chose another host in the database to handle the client connection, the client disconnects from the initial host. Otherwise, the client jumps to step 6.
5. The client establishes a connection to the host that will handle its connection. The client sets this second connection request so that the second host does not interpret the connection as a request for load balancing.
6. The client connection proceeds as usual, (negotiating encryption if the connection has SSL enabled, and proceeding to authenticating the user).

This process is transparent to the client application. The client driver automatically disconnects from the initial host and reconnects to the host selected for load balancing.

Requirements

- In mixed IPv4 and IPv6 environments, balancing only works for the address family for which you have configured native load balancing. For example, if you have configured load balancing using an IPv4 address, then IPv6 clients cannot use load balancing, however the IPv6 clients can still connect, but load balancing does not occur.
- The native load balancer returns an IP address for the client to use. This address must be one that the client can reach. If your nodes are on a private network, native load-balancing requires you to publish a public address. See `export_address` in the system table [NODES](#).

Load Balancing Schemes

The load balancing scheme controls how a host selects which host to handle a client connection. There are three available schemes:

- **NONE** (default): Disables native connection load balancing.
- **ROUNDROBIN**: Chooses the next host from a circular list of hosts in the cluster that are up—for example, in a three-node cluster, iterates over node1, node2, and node3, then wraps back to node1. Each host in the cluster maintains its own pointer to the next host in the circular list, rather than there being a single cluster-wide state.
- **RANDOM**: Randomly chooses a host from among all hosts in the cluster that are up.

You set the native connection load balancing scheme using the [SET_LOAD_BALANCE_POLICY](#) function. See [Enabling and Disabling Native Connection Load Balancing](#) for instructions.

Driver Notes

- Native connection load balancing works with the ADO.NET driver's connection pooling. The connection the client makes to the initial host, and the final connection to the load-balanced host, use pooled connections if they are available.
- If a client application uses the JDBC and ODBC driver with third-party connection pooling solutions, the initial connection is not pooled because it is not a full client connection. The final connection is pooled because it is a standard client connection.

Connection Failover

The client libraries include a failover feature that allow them to connect to backup hosts if the host specified in the connection properties is unreachable. When using native connection load balancing, this failover feature is only used for the initial connection to the database. If the host to which the client was redirected does not respond to the client's connection request, the client does not attempt to connect to a backup host and instead returns a connection error to the user.

Clients are redirected only to hosts that are known to be up. Thus, this sort of connection failure should only occur if the targeted host goes down at the same moment the client is redirected to it. For more information, see [ADO.NET Connection Failover](#), [JDBC Connection Failover](#), and [ODBC Connection Failover](#) in Connecting to Vertica.

Related Tasks

Enabling and Disabling Native Connection Load Balancing.....	1035
Monitoring Native Connection Load Balancing.....	1037
Enabling Native Connection Load Balancing in ODBC.....	4473
Enabling Native Connection Load Balancing in JDBC.....	4536
Enabling Native Connection Load Balancing in ADO.NET.....	4608

Reference Materials

RESET_LOAD_BALANCE_POLICY.....	2915
SET_LOAD_BALANCE_POLICY.....	2916
JDBC Connection Properties.....	4521
Data Source Name (DSN) Connection Properties.....	4455

Enabling and Disabling Native Connection Load Balancing

Only a database superuser can enable or disable native connection load balancing. To enable or disable load balancing, use the [SET_LOAD_BALANCE_POLICY](#) function to set the load balance policy. Setting the load balance policy to anything other than 'NONE' enables load balancing on the server. The following example enables native connection load balancing by setting the load balancing policy to ROUNDROBIN.

```
=> SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');
          SET_LOAD_BALANCE_POLICY
-----
Successfully changed the client initiator load balancing policy to: roundrobin
(1 row)
```

To disable native connection load balancing, use `SET_LOAD_BALANCE_POLICY` to set the policy to 'NONE':

```
=> SELECT SET_LOAD_BALANCE_POLICY('NONE');  
SET_LOAD_BALANCE_POLICY  
-----  
Successfully changed the client initiator load balancing policy to: none  
(1 row)
```

Note: When a client makes a connection, the native load-balancer chooses a node and returns the value from the `export_address` column in the `NODES` table. The client then uses the `export_address` to connect. The `node_address` specifies the address to use for inter-node and spread communications. When a database is installed, the `export_address` and `node_address` are set to the same value. If you installed Vertica on a private address, then you must set the `export_address` to a *public* address for each node.

By default, client connections are not load balanced, even when connection load balancing is enabled on the server. Clients must set a connection parameter to indicate they are willing to have their connection request load balanced. See [Enabling Native Connection Load Balancing in ADO.NET](#), [Enabling Native Connection Load Balancing in JDBC](#) and [Enabling Native Connection Load Balancing in ODBC](#) in [Connecting to Vertica](#) for more information.

Important: In mixed IPv4 and IPv6 environments, balancing only works for the address family for which you have configured load balancing. For example, if you have configured load balancing using an IPv4 address, then IPv6 clients cannot use load balancing, however the IPv6 clients can still connect, but load balancing does not occur.

Resetting the Load Balancing State

When the load balancing policy is `ROUNDROBIN`, each host in the Vertica cluster maintains its own state of which host it will select to handle the next client connection. You can reset this state to its initial value (usually, the host with the lowest-node id) using the `RESET_LOAD_BALANCE_POLICY` function:

```
=> SELECT RESET_LOAD_BALANCE_POLICY();  
RESET_LOAD_BALANCE_POLICY  
-----  
Successfully reset stateful client load balance policies: "roundrobin".  
(1 row)
```

Related Information

About Native Connection Load Balancing1033

Related Tasks

Monitoring Native Connection Load Balancing.....	1037
Enabling Native Connection Load Balancing in ODBC.....	4473
Enabling Native Connection Load Balancing in JDBC.....	4536
Enabling Native Connection Load Balancing in ADO.NET.....	4608

Reference Materials

RESET_LOAD_BALANCE_POLICY.....	2915
SET_LOAD_BALANCE_POLICY.....	2916
JDBC Connection Properties.....	4521
Data Source Name (DSN) Connection Properties.....	4455

Monitoring Native Connection Load Balancing

Query the `LOAD_BALANCE_POLICY` column of the `V_CATALOG.DATABASES` to determine the state of native connection load balancing on your server:

```
=> SELECT LOAD_BALANCE_POLICY FROM V_CATALOG.DATABASES;
LOAD_BALANCE_POLICY
-----
roundrobin
(1 row)
```

Determining to Which Node a Client Has Connected

A client can determine the node to which is has connected by querying the `NODE_NAME` column of the `V_MONITOR.CURRENT_SESSION` table:

```
=> SELECT NODE_NAME FROM V_MONITOR.CURRENT_SESSION;
NODE_NAME
-----
v_vmart_node0002
(1 row)
```

Related Information

About Native Connection Load Balancing.....	1033
---	------

Related Tasks

Enabling and Disabling Native Connection Load Balancing.....	1035
Enabling Native Connection Load Balancing in ODBC.....	4473
Enabling Native Connection Load Balancing in JDBC.....	4536
Enabling Native Connection Load Balancing in ADO.NET.....	4608

Reference Materials

RESET_LOAD_BALANCE_POLICY.....	2915
SET_LOAD_BALANCE_POLICY.....	2916
JDBC Connection Properties.....	4521
Data Source Name (DSN) Connection Properties.....	4455

Managing Nodes

Vertica provides the ability to [add](#), [remove](#), and [replace](#) nodes on a live cluster that is actively processing queries. This ability lets you scale the database without interrupting users.

In This Section

Stop Vertica on a Node

In some cases, you need to take down a node to perform maintenance tasks, or upgrade hardware.

1. Check the K-safety level of your cluster:

```
=> SELECT current_fault_tolerance FROM system;
current_fault_tolerance
-----
1
(1 row)
```

Stopping the node might require you to temporarily reduce the K-safety level of the database. For details, see [Lowering K-Safety to Enable Node Removal](#).

Caution: If you must reduce K-safety to 0, first [back up the database](#).

2. Run Administration Tools, select **Advanced Menu**, and click **OK**.
3. Select **Stop Vertica on Host** and click **OK**.
4. Choose the host that you want to stop and click **OK**.
5. Return to the Main Menu, select **View Database Cluster State**, and click **OK**. The host you previously stopped should appear **DOWN**.
6. You can now perform maintenance.

See [Restart Vertica on a Node](#) for details about restarting Vertica on a node.

Restart Vertica on a Node

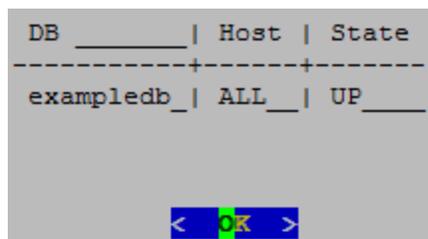
After [stopping a node](#) to perform maintenance, upgrade the hardware, or another similar task, you can bring the node back up. Performing this process reconnects the node with the database.

Restarting Vertica on a Node

1. Run Administration Tools. From the Main Menu select **Restart Vertica on Host** and click **OK**.
2. Select the database and click **OK**.
3. Select the host that you want to restart and click **OK**.

Note: This process may take a few moments.

4. Return to the Main Menu, select **View Database Cluster State**, and click **OK**. The host you restarted now appears as UP, as shown.



```
DB _____ | Host | State
-----+-----+-----
exampledb_ | ALL_ | UP_
```

< OK >

Setting Node Type

When you create a node, Vertica automatically sets its type to PERMANENT. This enables Vertica to use this node to store data. You can change a node's type with [ALTER NODE](#), to one of the following:

- **EPHEMERAL:** A node that is in transition from one type to another—typically, from PERMANENT to either STANDBY or EXECUTE.

- **STANDBY:** A node that is reserved to replace any node when it goes down. When used as a replacement node, Vertica changes its type to **PERMANENT**. A standby node stores no segments or data until it is called to replace a down node. At that time, Vertica changes its type to **PERMANENT**. For more information, see [Active Standby Nodes](#).
- **EXECUTE:** A node that is reserved for computation purposes only. An execute node contains no segments or data.

Active Standby Nodes

An *active standby node* is a specialized type of Vertica [node](#). An active standby node exists as a backup node, ready to replace a failed node. Unlike standard Vertica nodes, an active standby node does not perform computations or contain data. If a standard(permanent) node fails, an active standby node can replace the failed node, after the failed node exceeds the failover time limit. When it take the place of a failed node, the active standby node contains all of the projections and performs all of the calculations of the replaced node.

To deploy active standby nodes automatically, you must first configure the `FailoverToStandbyAfter` parameter. If possible, Vertica selects a standby node from the same fault group as the failed node. Otherwise, Vertica randomly selects an available active standby node.

If you are an administrator, you can manually replace a failed node using the [ALTER NODE](#) command. You can specify a particular standby node to replace a failed node, or you can allow Vertica to choose a node. As with automatic node replacement, Vertica defaults to a standby node from the same fault group as the failed node. If the fault group has no available standby nodes, Vertica selects any available active standby node.

In This Section

Creating an Active Standby Node

You can create active standby nodes at the same time that you create your database or at a later time.

Note: When you create an active standby node, be sure to add any necessary storage locations. For more information, refer to [Adding Storage Locations](#).

Creating an Active Standby Node with a New Database

1. [Create a database](#), including the nodes that you intend to use as active standby nodes.
2. Using vsql, connect to a node **other than** the node that you want to use as an active standby node.
3. Use the [ALTER NODE](#) SQL statement to convert the node from a permanent node to an active standby node. The following example shows a typical alter node command.

```
=> ALTER NODE v_mart_node5 STANDBY;
```

Once you issue the ALTER NODE statement, the affected node goes down and restarts as an active standby node.

Creating an Active Standby Node with an Existing Database

When you are creating a new node with the intent of making it into an active standby node, shift the new node to ephemeral status as quickly as possible to prevent the cluster from moving data to the new node.

1. [Add a new node to your database](#). Do not rebalance the database.
2. Using vsql, connect to a node **other than** the node that you want to use as an active standby node.
3. Use the [ALTER NODE](#) SQL statement to convert the node from a permanent node to an ephemeral node. For example, the following command would set v_mart_node5 to a [node type](#) of ephemeral.

```
=> ALTER NODE v_mart_node5 EPHEMERAL;
```

4. [Rebalance the cluster](#) to move any data from the ephemeral node.
5. Use the [ALTER NODE](#) SQL statement to convert the node from an ephemeral node to an active standby node. The following example shows v_mart_node5 moving from ephemeral status to standby status.

```
=> ALTER NODE v_mart_node5 STANDBY;
```

Replace a Node Using an Active Standby Node

Note: A node must be down for Vertica before you can replace it with an active standby node. If you attempt to replace a functioning node, Vertica displays an error message.

Replacing a Node with Automatic Failover

Vertica can automatically replace a failed node with an active standby node. In the [General Parameters](#) configure the `FailoverToStandbyAfter` parameter to allow automatic failover.

Manually Replacing a Node

1. Connect to the database with [Administration Tools](#) or `vsql`.
2. Enter the command: `ALTER NODE node-name REPLACE WITH standby-node-name;`

Revert from an Active Standby Node

If an active standby node has replaced a Vertica node in your cluster, you can revert from the active standby node on a node-by-node basis or on a cluster-wide basis.

To revert from a single active standby node:

1. Connect to the database with [Administration Tools](#) or via `vsql`.
2. Enter the command: `ALTER NODE (node-name) RESET;`

To revert all cluster nodes from active standby:

1. Connect to the database with [Administration Tools](#) or via `vsql`.
2. Enter the command: `ALTER DATABASE (database-name) RESET STANDBY;`

If a down node cannot resume operation, Vertica ignores the reset request and leaves the standby node in place.

Fault Groups

Fault groups let you configure Vertica for your physical cluster layout. Sharing your cluster topology allows you to use Terrace Routing to reduce the buffer requirements of large queries and helps minimize the risk of correlated failures inherent in your environment, usually caused by shared resources.

Vertica automatically creates fault groups around control nodes (servers that run spread) in large cluster arrangements, placing nodes that share a control node in the same fault group. Automatic and user-defined fault groups do not include ephemeral nodes because such nodes hold no data.

Consider defining your own fault groups specific to your cluster's physical layout if you want to:

- Use Terrace Routing to reduce the buffer requirements of large queries.
- Reduce the risk of correlated failures. For example, by defining your rack layout, Vertica could tolerate a rack failure.
- Influence the placement of control nodes in the cluster.

Vertica supports complex, hierarchical fault groups of different shapes and sizes. The database platform provides a fault group script (DDL generator), SQL statements, system tables, and other monitoring tools.

See [High Availability With Fault Groups](#) for an overview of fault groups with a cluster topology example.

See Also

- [Creating a Fault Group Input File](#)
- [Creating Fault Groups Using the Fault Group Script](#)
- [Dropping Fault Groups](#)
- [Monitoring Fault Groups](#)

About the Fault Group Script

To help you define fault groups on your cluster, Vertica provides a script named `fault_group_ddl_generator.py` in the `/opt/vertica/scripts` directory. This script generates the SQL statements you need to run to create fault groups.

The `fault_group_ddl_generator.py` script does not create fault groups for you, but you can copy the output to a file. Then, when you run the helper script, you can use `\i` or `vsqf-f` commands to pass the cluster topology to Vertica.

The fault group script takes the following arguments:

- The database name
- The fault group input file

For example:

```
$ python /opt/vertica/scripts/fault_group_ddl_generator.py VMartdb fault_grp_input.out
```

See Also

- [Creating a Fault Group Input File](#)
- [Creating Fault Groups Using the Fault Group Script](#)
- [Dropping Fault Groups](#)
- [Monitoring Fault Groups](#)
- [Fault Groups](#)

Creating a Fault Group Input File

Use a text editor to create a fault group input file for the targeted cluster.

The following example shows how you can create a fault group input file for a cluster that has 8 racks with 8 nodes on each rack—for a total of 64 nodes in the cluster.

1. On the first line of the file, list the parent (top-level) fault groups, delimited by spaces.

```
rack1 rack2 rack3 rack4 rack5 rack6 rack7 rack8
```

2. On the subsequent lines, list the parent fault group followed by an equals sign (=). After the equals sign, list the nodes or fault groups delimited by spaces.

```
<parent> = <child_1> <child_2> <child_n...>
```

Such as:

```
rack1 = v_vmart_node0001 v_vmart_node0002 v_vmart_node0003 v_vmart_node0004  
rack2 = v_vmart_node0005 v_vmart_node0006 v_vmart_node0007 v_vmart_node0008  
rack3 = v_vmart_node0009 v_vmart_node0010 v_vmart_node0011 v_vmart_node0012  
rack4 = v_vmart_node0013 v_vmart_node0014 v_vmart_node0015 v_vmart_node0016  
rack5 = v_vmart_node0017 v_vmart_node0018 v_vmart_node0019 v_vmart_node0020  
rack6 = v_vmart_node0021 v_vmart_node0022 v_vmart_node0023 v_vmart_node0024  
rack7 = v_vmart_node0025 v_vmart_node0026 v_vmart_node0027 v_vmart_node0028  
rack8 = v_vmart_node0029 v_vmart_node0030 v_vmart_node0031 v_vmart_node0032
```

After the first row of parent fault groups, the order in which you write the group descriptions does not matter. All fault groups that you define in this file must refer back to a parent fault group. You can indicate the parent group directly or by specifying the child of a fault group that is the child of a parent fault group.

Such as:

```
rack1 rack2 rack3 rack4 rack5 rack6 rack7 rack8  
rack1 = v_vmart_node0001 v_vmart_node0002 v_vmart_node0003 v_vmart_node0004  
rack2 = v_vmart_node0005 v_vmart_node0006 v_vmart_node0007 v_vmart_node0008  
rack3 = v_vmart_node0009 v_vmart_node0010 v_vmart_node0011 v_vmart_node0012  
rack4 = v_vmart_node0013 v_vmart_node0014 v_vmart_node0015 v_vmart_node0016  
rack5 = v_vmart_node0017 v_vmart_node0018 v_vmart_node0019 v_vmart_node0020  
rack6 = v_vmart_node0021 v_vmart_node0022 v_vmart_node0023 v_vmart_node0024  
rack7 = v_vmart_node0025 v_vmart_node0026 v_vmart_node0027 v_vmart_node0028  
rack8 = v_vmart_node0029 v_vmart_node0030 v_vmart_node0031 v_vmart_node0032
```

After you create your fault group input file, you are ready to run the `fault_group_ddl_generator.py`. This script generates the DDL statements you need to create fault groups in Vertica.

If your Vertica database is co-located on a Hadoop cluster, and that cluster uses more than one rack, you can use fault groups to improve performance. See [Configuring Rack Locality](#).

See Also

[Creating Fault Groups Using the Fault Group Script](#)

Creating Fault Groups Using the Fault Group Script

When you define fault groups, Vertica distributes data segments across the cluster. This allows the cluster to be aware of your cluster topology so it can tolerate correlated failures inherent in your environment, such as a rack failure. For an overview, see [High Availability With Fault Groups](#).

Prerequisites for Defining Fault Groups

Defining fault groups requires careful and thorough network planning. You must have a solid understanding of your network topology.

Important: The user who creates fault groups must be a superuser.

Before you can define fault groups, you must:

- A fault group input file. For more information, see [Creating a Fault Group Input File](#).
- An already-existing database

Run the Fault Group Script

1. As the database administrator, run the `fault_group_ddl_generator.py` script and include the following arguments:
 - The database name
 - The fault group input file

For example:

```
$ python /opt/vertica/scripts/fault_group_ddl_generator.py  
VMart fault_groups_VMart.out > fault_group_ddl.sql
```

The preceding command writes the output of `fault_group_ddl_generator.py` to a file (`fault_group_ddl.sql`). This approach allows you to run a single SQL script instead of multiple DDL statements. Also consider saving your input file so you can modify fault groups later, such as after you expand the cluster or change the distribution of control nodes.

- Using `vsql`, run the DDL statements in `fault_group_ddl.sql` or execute the commands in the file using `vsql`.

```
=> \i fault_group_ddl.sql
```

- If you have large cluster enabled, realign the control nodes, as shown. Otherwise, proceed to the next step.

```
=> SELECT REALIGN_CONTROL_NODES();
```

- Save cluster changes to the spread configuration file.

```
=> SELECT RELOAD_SPREAD(true);
```

- Use Administration Tools to restart the database.
- Save changes to the cluster's data layout by calling the `REBALANCE_CLUSTER` function.

```
=> SELECT REBALANCE_CLUSTER();
```

See Also

- [Cluster Management Functions](#)
- [Terrace Routing](#)
- [CREATE FAULT GROUP](#)
- [ALTER FAULT GROUP](#)
- [DROP FAULT GROUP](#)
- [ALTER DATABASE](#)

Monitoring Fault Groups

You can monitor fault groups by querying Vertica system tables or by logging in to the Management Console (MC) interface.

Monitor Fault Groups Using System Tables

Use the following system tables to view information about fault groups and cluster vulnerabilities, such as the nodes the cluster cannot lose without the database going down:

- [V_CATALOG.FAULT_GROUPS](#)—View the hierarchy of all fault groups in the cluster.
- [V_CATALOG.CLUSTER_LAYOUT](#)—Observe the arrangement of the nodes participating in the data business and the fault groups that affect them. Ephemeral nodes do not appear in the cluster layout ring because they hold no data.

Monitoring Fault Groups Using Management Console

An MC administrator can monitor and highlight fault groups of interest by following these steps:

1. Click the running database you want to monitor and click **Manage** in the task bar.
2. Open the **Fault Group View** menu, and select the fault groups you want to view.
3. (Optional) Hide nodes that are not in the selected fault group to focus on fault groups of interest.

Nodes assigned to a fault group each have a colored bubble attached to the upper-left corner of the node icon. Each fault group has a unique color. If the number of fault groups exceeds the number of colors available, MC recycles the colors used previously.

Because Vertica supports complex, hierarchical fault groups of different shapes and sizes, MC displays multiple fault group participation as a stack of different-colored bubbles. The higher bubbles represent a lower-tiered fault group, which means that bubble is closer to the parent fault group, not the child or grandchild fault group.

For more information about fault group hierarchy, see [High Availability With Fault Groups](#).

Dropping Fault Groups

When you remove a fault group from the cluster, be aware that the drop operation removes the specified fault group and its child fault groups. Vertica places all nodes under the parent of the dropped fault group. To see the current fault group hierarchy in the cluster, query the [FAULT_GROUPS](#) system table.

Drop a Fault Group

Use the `DROP FAULT GROUP` statement to remove a fault group from the cluster. The following example shows how you can drop the `group2` fault group:

```
=> DROP FAULT GROUP group2;  
DROP FAULT GROUP
```

Drop All Fault Groups

Use the `ALTER DATABASE` statement to drop all fault groups, along with any child fault groups, from the specified database cluster.

The following command drops all fault groups from the `vmartdb` database.

```
=> ALTER DATABASE exampledb DROP ALL FAULT GROUP;  
ALTER DATABASE
```

Add Nodes Back to a Fault Group

To add a node back to a fault group, you must manually reassign it to a new or existing fault group. To do so, use the `CREATE FAULT GROUP` and `ALTER FAULT GROUP..ADD NODE` statements.

See Also

- [DROP FAULT GROUP](#)
- [CREATE FAULT GROUP](#)
- [ALTER FAULT GROUP..ADD NODE](#)
- [Creating Fault Groups Using the Fault Group Script](#)
- [About the Fault Group Script](#)
- [Creating a Fault Group Input File](#)

Large Cluster

To support scaling of existing clusters into large clusters and improve control message performance, Vertica delegates control message responsibilities to a subset of Vertica nodes, called **control nodes**. Control nodes communicate with each other. Other cluster nodes are assigned to a control node, which they use for control message communications.

Control nodes on large clusters

On clusters of 120 or more nodes, a large cluster layout is necessary and enabled by default. Vertica makes automatic control node assignments unless you use one of the following options:

If you want to ...	Do this
Install a new cluster before you create a database	Run the Vertica installation script with the <code>--large-cluster <integer></code> arguments. See the following topics for details: <ul style="list-style-type: none">• Installing Vertica with the Installation Script in Installing Vertica• Installing a Large Cluster in this guide
<ul style="list-style-type: none">• Expand an existing cluster for pre-existing databases• Change control nodes on an existing cluster	Use cluster management functions described in Defining and Realigning Control Nodes on an Existing Cluster .
Influence the placement of control nodes in your cluster's physical layout	Define fault groups to configure Vertica for your cluster. See Fault Groups for details.

Control nodes on small clusters

If your cluster has fewer than 120 nodes, large cluster is neither necessary nor automatically applied. As a result, all nodes are control nodes. However, Vertica lets you define control nodes on any sized cluster. Some environments, such as cloud deployments that might have higher network latency, could benefit from a smaller number of control nodes.

For details, see [Planning a Large Cluster Arrangement](#) and [Installing a Large Cluster](#).

Planning a Large Cluster

In a large cluster layout of 120 nodes or more, nodes form a correlated failure group, governed by their control node—the node that runs control messaging (spread). If a control node fails, all nodes in its host group also fail.

This topic provides tips on how to plan for a large cluster arrangement. See [Installing a Large Cluster](#) and [Large Cluster Best Practices](#) for more information.

Planning the number of control nodes

Configuring a large cluster requires careful and thorough network planning. You must have a solid understanding of your network topology before you configure the cluster.

To assess how many cluster nodes should be control nodes, use the square root of the total number of nodes expected to be in the database cluster to help satisfy both data K-Safety and rack fault tolerance for the cluster. Depending on the result, you might need to adjust the number of control nodes to account for your physical hardware/rack count. For example, if you have 121 nodes (with a result of 11), and your nodes will be distributed across 8 racks, you might want to increase the number of control nodes to 16 so you have two control nodes per rack.

Specifying the number of control nodes

Vertica provides different tools to help you define the number of control nodes, depending on your current configuration. Consider the following scenarios, in which cluster nodes are distributed among three racks in different configurations:

If you cluster fits this scenario ...	Consider this setup
Three control nodes. All other nodes are evenly distributed among the three racks.	Specify one control node per rack.
Five control nodes and three racks.	Specify two control nodes on two racks each and one control node on the final rack.
Four control nodes. One rack has twice as many nodes as the other racks.	Specify two control nodes on the larger rack and one control node each on the other two racks.

Installing a Large Cluster

Whether you are forming a new large cluster (adding all nodes for the first time) or expanding an existing cluster to a large cluster, Vertica provides two methods that let you specify the number of control nodes (the nodes that run control messaging). See the following sections for details:

- [If you want to install a new large cluster](#)
- [If you want to expand an existing cluster](#)

If you want to install a new large cluster

To configure Vertica for a new, large cluster, pass the `install_vertica` script the `--large-cluster <integer>` argument. Vertica selects the first `<integer>` number of hosts from the comma-separated `--hosts host_list` as control nodes and assigns all other hosts you specify in the `--hosts` argument to a control node based on a round-robin model.

Note: The number of hosts you include in the `--hosts` argument determines a large cluster layout, not the number of nodes you later include in the database. If you specify 120 or more hosts in the `--hosts` list, but you do not specifically enable large cluster by providing the `--large-cluster` argument, Vertica automatically enables large cluster and configures control nodes for you.

To help control nodes and the nodes assigned to them be configured for the highest possible fault tolerance, you must specify hosts in the `--hosts host_list` in a specific order. For example, if you have four sets of hosts on four racks, the first four nodes in the `--hosts host_list` must be one host from each rack in order to have one control node per rack. Then the list must consist of four hosts from each rack in line with the first four hosts. You'll continue to use this pattern of host listing for all targeted hosts. See [Sample rack-based cluster hosts topology](#) below for examples.

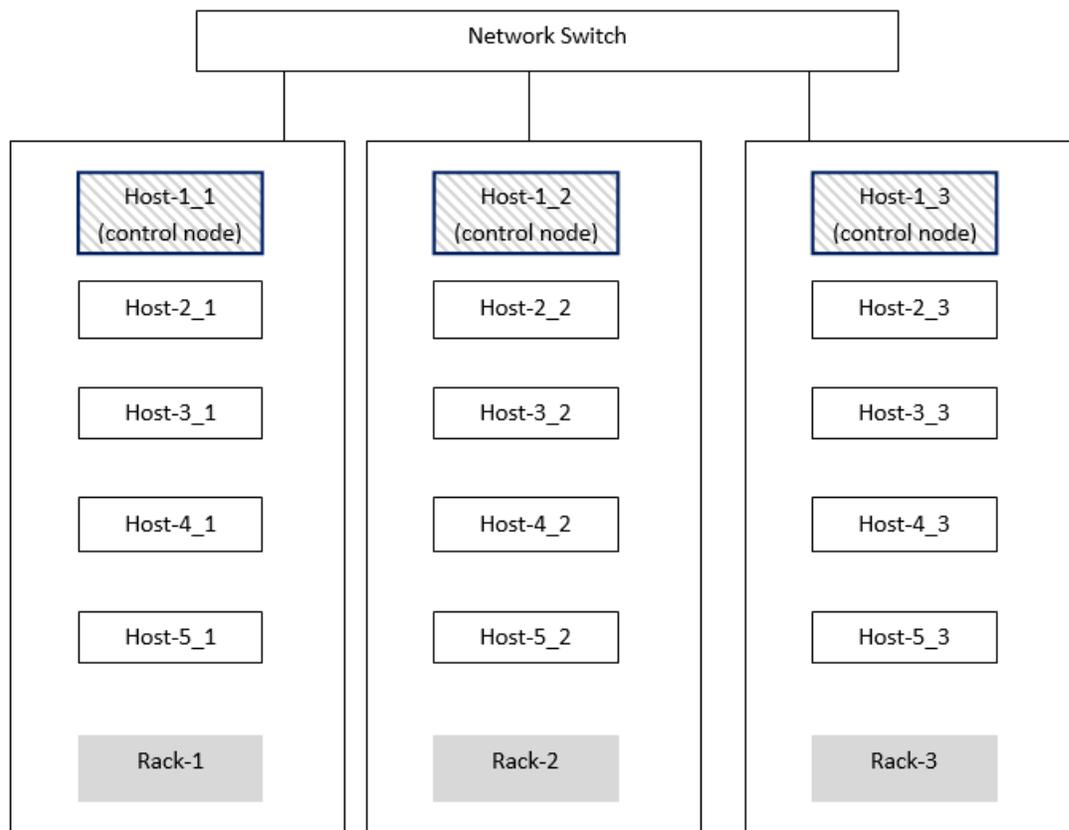
Tip: If you pass the `--large-cluster` argument a DEFAULT value instead of an `<integer>` value, Vertica calculates a number of control nodes based on the total number of nodes specified in the `--hosts host_list` argument. If you want a specific number of control nodes on the cluster, you must use the `<integer>` value.

For more information, see the following topics:

- [Planning a Large Cluster Arrangement](#)
- [Installing Vertica with the Installation Script](#) in Installing Vertica

Sample rack-based cluster hosts topology

This example shows a simple, multi-rack cluster layout, in which cluster nodes are evenly distributed across three racks. Each rack has one control node.



In the rack-based example:

- Rack-1, Rack-2, and Rack-3 are managed by a single network switch
- Host-1_1, Host-1_2, and Host-1_3 are control nodes
- All hosts on Rack-1 are assigned to control node Host-1_1
- All hosts on Rack-2 are assigned to control node Host-1_2
- All hosts on Rack-3 are assigned to control node Host-1_3

In the following `install_vertica` script fragment, note the order of the hosts in the `--hosts` list argument. The final arguments specifically enable large cluster and provide the number of control nodes (3):

```
... install_vertica --hosts Host-1-1,Host-1-2,Host-1-3,  
Host-2-1,Host-2-2,Host-2-3,Host-3-1,Host-3-2,Host-3-3,  
Host-4-1,Host-4-2,Host-4-3,Host-5-1,Host-5-2,Host-5-3 -rpm  
<vertica-package-name> <other-required-options>  
--large-cluster 3
```

After the installation process completes, use the Administration Tools to create a database. This operation generates a Vertica cluster with three control nodes and their respective associated hosts that reside on the same racks as the control node.

If you want to expand an existing cluster

When you add a node to an existing cluster, Vertica places the new node in an appropriate location within the cluster ring. Vertica then assigns the newly-added node to a control node, based on the cluster's current allocations.

To give you more flexibility and control over which nodes run Spread, you can use the `SET_CONTROL_SET_SIZE(integer)` function. This function works like the installation script's `--large-cluster <integer>` option. See [Defining and Realigning Control Nodes on an Existing Cluster](#) for details.

Important: The Vertica installation script cannot alter the database cluster.

Defining and Realigning Control Nodes on an Existing Cluster

This topic describes how to set up or change control node assignments on an existing cluster using a series of [cluster management functions](#). It assumes you already know how many control nodes the cluster needs for failover safety. See [Planning a Large Cluster Arrangement](#) for more information.

Note: If you are adding control nodes for the first time, run the Vertica installation script using the `--large-cluster <integer>` argument. See [Installing Vertica with the Installation Script](#) in [Installing Vertica](#).

Setting up control nodes on an existing cluster makes the following changes to the cluster:

- Configures the number of nodes that run spread.
- Assigns each non-control cluster node to a control node.

- Saves the new layout to the spread configuration file.
- Redistributes data across the cluster to improve fault tolerance.

How to set up control nodes on an existing cluster

After you add, remove, or swap nodes on an existing cluster, perform the following steps. This procedure helps the cluster maintain adequate control messaging distribution for failover safety. For more details, see "Control node assignment/realignment" in [Large Cluster Best Practices](#).

1. As the database administrator, log in to the Administration Tools and connect to the database.
2. Call the `SET_CONTROL_SET_SIZE(integer)` function with an integer argument that specifies the number of control nodes you want. For example, 4:

```
=> SELECT SET_CONTROL_SET_SIZE(4);
```

3. Call the `REALIGN_CONTROL_NODES()` function without arguments:

```
=> SELECT REALIGN_CONTROL_NODES();
```

4. Call the `RELOAD_SPREAD(true)` function to save changes to the spread configuration file:

```
=> SELECT RELOAD_SPREAD(true);
```

5. After the `RELOAD_SPREAD()` operation finishes, log back in to the Administration Tools, and restart the database.
6. Call the `REBALANCE_CLUSTER()` function to distribute data across the cluster:

```
=> SELECT REBALANCE_CLUSTER();
```

Important: You must run `REBALANCE_CLUSTER` for fault tolerance to be realized.

For more details about the functions used in this procedure, see [Cluster Management Functions](#) in the SQL Reference Manual.

Expanding the Database to a Large Cluster

If you have an existing database cluster that you want to expand to a large cluster (more than 120 nodes), follow these steps:

1. Log in to the Administration Tools as the database administrator and stop the database.
2. As root or a user with sudo privileges, open a BASH shell and run the `install_vertica` script with the `--add-hosts` argument, providing a comma-separated list of hosts you want to add to an existing Vertica cluster. See [Installing Vertica with the Installation Script](#) in [Installing Vertica](#).
3. Exit the shell and re-establish a `vsq` connection as the database administrator.
4. Log in to the Administration Tools and start the database.
5. Use the Administration Tools **Advanced Menu > Cluster Management > Add Hosts** option to add the standby hosts you created in Step 2.
6. Run `SET_CONTROL_SET_SIZE(integer)` to specify the number of control nodes you want on the cluster. See [Defining and Realigning Control Nodes on an Existing Cluster](#).
7. Optionally create fault groups to further define the layout of the control nodes within the physical cluster. See [Fault Groups](#).

Terrace Routing

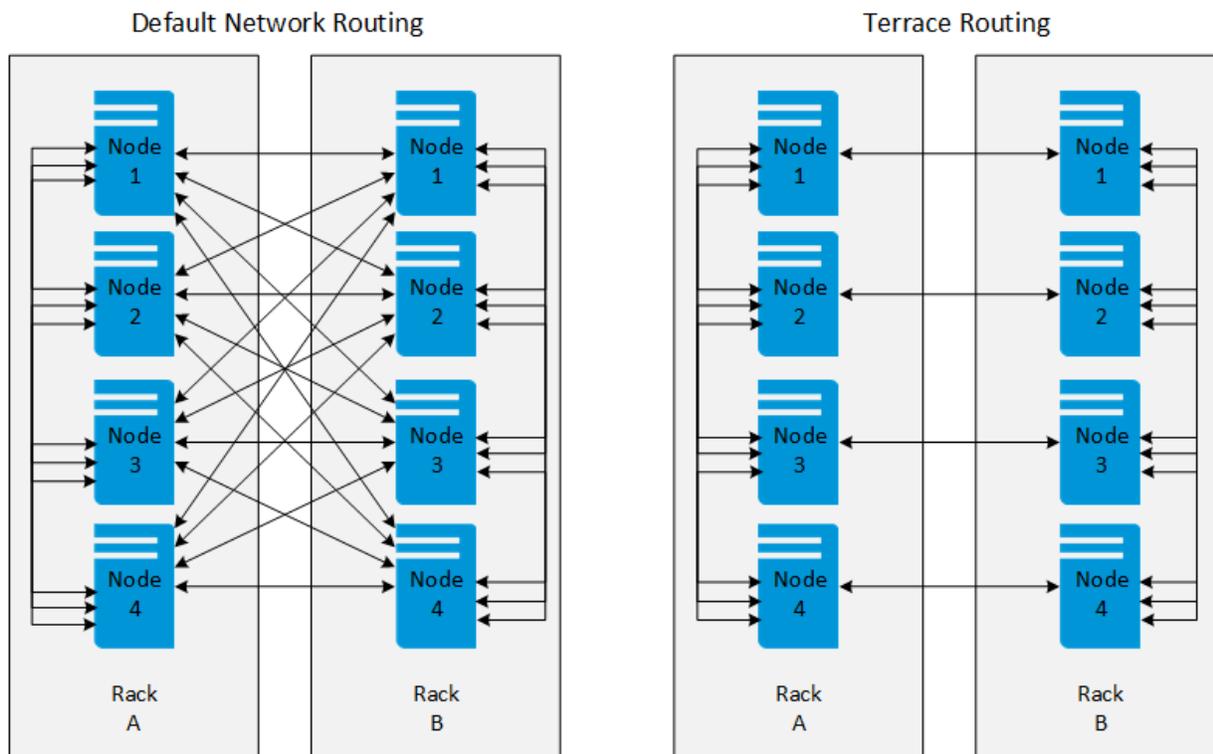
By default, nodes in a Vertica cluster form a fully connected network (complete graph). Therefore, large Vertica clusters contain an extremely large number of connections between nodes. Each connection requires buffering. Thus, your current system can have many connections, causing queries to have extremely large data requirements on each node.

Terrace routing is a feature that can reduce the buffer requirements of large queries. Use terrace routing in situations where you have large queries and clusters with a large number of nodes. Without terrace routing, these situations would otherwise require excessive buffer space.

How Terrace Routing Changes Routing Behavior

In the case when data needs to be sent from each node across an entire network. In this case, Vertica uses the n^{th} node in each rack to send to all the nodes in all the racks. Terrace routing

changes this behavior. Instead of sending to all nodes in all racks, the n^{th} node in each rack sends to the n^{th} nodes in all the other racks. It additionally sends to all the nodes in its own rack.



Before Enabling Terrace Routing

Before you apply terrace routing to your database make sure you have read the following topics and completed the tasks described:

- [Fault Groups](#)
- [Creating a Fault Group Input File](#)
- [Creating Fault Groups Using the Fault Group Script](#)
- [High Availability with Fault Groups](#)

How to Apply Terrace Routing

Your Vertica Analytics Platform is not aware of the topology of the cluster on which your Vertica database is running. By applying fault groups to your database, you can make your Vertica Analytics Platform aware of your network's topology.

Implementing terrace routing requires use of fault groups. Terrace routing uses the fault-group definitions to represent the topology of the network of the cluster, because rack membership

is a natural division for fault groups. In a terrace-routing approach, Vertica first distributes data within the rack (fault group) before forwarding it between racks. For information about creating fault groups, see [Creating Fault Groups Using the Fault Group Script](#).

For this reason, terrace routing uses the network within the rack to reduce the buffering requirements for queries. Networks between racks can have a higher capacity so there is a performance tradeoff when a connection goes off-rack.

Calculate the TerraceRoutingFactor

TerraceRoutingFactor is a general parameter that sets the ratio of connections without terrace routing to the number of connections with terrace routing.

You can calculate the TerraceRoutingFactor value using this formula:

$$((RackCount * RackPopulation) - 1) / (RackCount + RackPopulation - 2) = TerraceRoutingFactor$$

RackCount is the number of racks or fault groups in your cluster, and *RackPopulation* is the number of nodes in each rack or fault group.

Enable Terrace Routing

The TerraceRoutingFactor is set to a default value of 1000.0, which is large enough to disable the parameter for even the largest clusters. Micro Focus recommends enabling terrace routing when your cluster contains 64 or more nodes or if your queries require excessive buffer space.

The following examples show how you can enable terrace routing for varying node and rack configurations and then disable it.

Set the TerraceRoutingFactor on a 64 node cluster with 8 racks and 8 nodes in each rack:

```
=> ALTER DATABASE MyDB SET TerraceRoutingFactor = 4.5;
```

Set the TerraceRoutingFactor on a 80 node cluster with 10 racks and 8 nodes in each rack in the current session:

```
=> ALTER SESSION SET TerraceRoutingFactor = 4.94;
```

Disable terrace routing:

```
=> ALTER DATABASE MyDB SET TerraceRoutingFactor = 1000;
```

Monitoring Large Clusters

Monitor large cluster traits by querying the following system tables:

- [V_CATALOG.LARGE_CLUSTER_CONFIGURATION_STATUS](#)—Shows the current spread hosts and the control designations in the catalog so you can see if they match.
- [V_MONITOR.CRITICAL_HOSTS](#)—Lists the hosts whose failure would cause the database to become unsafe and force a shutdown.

Tip: The `CRITICAL_HOSTS` view is especially useful for large cluster arrangements. For non-large clusters, query the [CRITICAL_NODES](#) table.

You might also want to query the following system tables:

- [V_CATALOG.FAULT_GROUPS](#)—Shows fault groups and their hierarchy in the cluster.
- [V_CATALOG.CLUSTER_LAYOUT](#)—Shows the relative position of the actual arrangement of the nodes participating in the database cluster and the fault groups that affect them.

Large Cluster Best Practices

Keep the following best practices in mind when you are planning and managing a large cluster implementation.

Planning the number of control nodes

To assess how many cluster nodes should be control nodes, use the square root of the total number of nodes expected to be in the database cluster to help satisfy both data K-Safety and rack fault tolerance for the cluster. Depending on the result, you might need to adjust the number of control nodes to account for your physical hardware/rack count. For example, if you have 121 nodes (with a result of 11), and your nodes will be distributed across 8 racks, you might want to increase the number of control nodes to 16 so you have two control nodes per rack.

See [Planning a Large Cluster Arrangement](#).

Control node assignment/realignment

After you specify the number of control nodes, you must update the control host's (spread) configuration files to reflect the catalog change. Certain [cluster management functions](#) might require that you run other functions or restart the database or both.

If, for example, you drop a control node, cluster nodes that point to it are reassigned to another control node. If that node fails, all the nodes assigned to it also fail, so you need to use

the Administration Tools to restart the database. In this scenario, you'd call the `REALIGN_CONTROL_NODES()` and `RELOAD_SPREAD(true)` functions, which notify nodes of the changes and realign fault groups. Calling `RELOAD_SPREAD(true)` connects an existing cluster node to a newly-assigned control node.

On the other hand, if you run `REALIGN_CONTROL_NODES()` multiple times in a row, the layout does not change beyond the initial setup, so you don't need to restart the database. But if you add or drop a node and then run `REALIGN_CONTROL_NODES()`, the function call could change many node assignments.

Here's what happens with control node assignments when you add or drop nodes, whether those nodes are control nodes or non-control nodes:

- **If you add a cluster node**—Vertica assigns a control node to the newly-added node based on the current cluster configuration. If the new node joins a fault group, it is assigned to a control node from that fault group and requires a database restart to reconnect to that control node. See [Fault Groups](#) for more information.
- **If you drop a non-control node**—Vertica quietly drops the cluster node. This operation could change the cluster and spread layout, so you must call `REBALANCE_CLUSTER()` after you drop a node.
- **If you drop a control node**—All nodes assigned to the control node go down. In large cluster implementations, however, the database remains up because the down nodes are not buddies with other cluster nodes.

Dropping a control node results in $(n-1)$ control nodes. You must call `REALIGN_CONTROL_NODES()` to reset the cluster so it has n control nodes, which might or might not be the same number as before you dropped the control node. Remaining nodes are assigned new control nodes. In this operation, Vertica makes control node assignments based on the cluster layout. When it makes the new assignments, it respects user-defined fault groups, if any, which you can view by querying the `V_CATALOG.CLUSTER_LAYOUT` system table, a view that also lets you see the proposed new layout for nodes in the cluster. If you want to influence the layout of control nodes in the cluster, you should define fault groups.

For more information, see [Defining and Realigning Control Nodes on an Existing Cluster](#) and [Rebalancing Data Across Nodes](#).

Allocate standby nodes

Have as many standby nodes available as you can, ideally on racks you are already using in the cluster. If a node suffers a non-transient failure, use the Administration Tools "Replace Host" utility to swap in a standby node.

Standby node availability is especially important for control nodes. If you are swapping a node that's a control node, all nodes assigned to the control node's host grouping will need to be taken offline while you swap in the standby node. For details on node replacement, see [Replacing Nodes](#).

Plan for cluster growth

If you plan to expand an existing cluster to 120 or more nodes, you can configure the number of control nodes for the cluster after you add the new nodes. See [Defining and Realigning Control Nodes](#).

Write custom fault groups

When you deploy a large cluster, Vertica automatically creates fault groups around control nodes, placing nodes that share a control node into the same fault group. Alternatively, you can specify which cluster nodes should reside in a particular correlated failure group and share a control node. See [High Availability With Fault Groups](#) in Vertica Concepts.

Use segmented projections

On large-cluster setups, minimize the use of unsegmented projections in favor of segmented projections. When you use segmented projections, Vertica creates buddy projections and distributes copies of segmented projections across database nodes. If a node fails, data remains available on the other cluster nodes.

Use the Database Designer

Micro Focus recommends that you use the Database Designer to create your physical schema. If you choose to design projections manually, you should segment large tables across all database nodes and replicate (unsegment) small table projections on all database nodes.

Elastic Cluster

You can scale your cluster up or down to meet the needs of your database. The most common case is to add nodes to your database cluster to accommodate more data and provide better query performance. However, you can scale down your cluster if you find that it is overprovisioned or if you need to divert hardware for other uses.

You scale your cluster by adding or removing nodes. Nodes can be added or removed without having to shut down or restart the database. After adding a node or before removing a node,

Vertica begins a rebalancing process that moves data around the cluster to populate the new nodes or move data off of nodes about to be removed from the database. During this process, data can be exchanged between nodes that are not being added or removed to maintain robust intelligent K-safety. If Vertica determines that the data cannot be rebalanced in a single iteration due to a lack of disk space, then the rebalance is done in multiple iterations.

To help make data rebalancing due to cluster scaling more efficient, Vertica locally segments data storage on each node so it can be easily moved to other nodes in the cluster. When a new node is added to the cluster, existing nodes in the cluster give up some of their data segments to populate the new node and exchange segments to keep the number of nodes that any one node depends upon to a minimum. This strategy keeps to a minimum the number of nodes that may become critical when a node fails (see [Critical Nodes/K-Safety](#)). When a node is being removed from the cluster, all of its storage containers are moved to other nodes in the cluster (which also relocates data segments to minimize nodes that may become critical when a node fails). This method of breaking data into portable segments is referred to as elastic cluster, since it makes enlarging or shrinking the cluster easier.

The alternative to elastic cluster is to resegment all of the data in the projection and redistribute it to all of the nodes in the database evenly any time a node is added or removed. This method requires more processing and more disk space, since it requires all of the data in all projections to essentially be dumped and reloaded.

Elastic Cluster Scaling Factor

In a new installation, each node has a *scaling factor* that specifies the number of local segments (see [Scaling Factor](#)). Rebalance efficiently redistributes data by relocating local segments provided that, after nodes are added or removed, there are sufficient local segments in the cluster to redistribute the data evenly (determined by [MAXIMUM_SKEW_PERCENT](#)). For example, if the scaling factor = 8, and there are initially 5 nodes, then there are a total of 40 local segments cluster wide.

If you add two additional nodes (7 nodes) Vertica relocates 5 local segments on 2 nodes, and 6 such segments on 5 nodes, resulting in roughly a 16.7% skew. Rebalance chooses relocates local segments only if the resulting skew is less than the allowed threshold, as determined by [MAXIMUM_SKEW_PERCENT](#). Otherwise, segmentation space (and hence data, if uniformly distributed over this space) is evenly distributed among the 7 nodes, and new local segment boundaries are drawn for each node, such that each node again has 8 local segments.

Note: By default, the scaling factor only has an effect while Vertica rebalances the database. While rebalancing, each node breaks the projection segments it contains into storage containers, which it then moves to other nodes if necessary. After rebalancing, the data is recombined into ROS containers. It is possible to have Vertica always group data

into storage containers. See [Local Data Segmentation](#) for more information.

Enabling and Disabling Elastic Cluster

You enable and disable elastic cluster using functions. See the entries for the [ENABLE_ELASTIC_CLUSTER](#) and [DISABLE_ELASTIC_CLUSTER](#) functions in the SQL Reference Manual.

Query the [ELASTIC_CLUSTER](#) system table to determine if elastic cluster is enabled:

```
=> SELECT is_enabled FROM ELASTIC_CLUSTER;
is_enabled
-----
t
(1 row)
```

Scaling Factor

To avoid an increased number of ROS containers, do not disable local segmentation and do not change the scaling factor.

Viewing Scaling Factor Settings

To view the scaling factor, query the [ELASTIC_CLUSTER](#) table:

```
=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
scaling_factor
-----
4
(1 row)

=> SELECT SET_SCALING_FACTOR(6);
SET_SCALING_FACTOR
-----
SET
(1 row)

=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
scaling_factor
-----
6
(1 row)
```

Setting the Scaling Factor

The scaling factor determines the number of storage containers that Vertica uses to store each projection across the database during rebalancing when local segmentation is enabled. When setting the scaling factor, follow these guidelines:

- The number of storage containers should be greater than or equal to the number of partitions multiplied by the number of local segments:

$$\text{num-storage-containers} \geq (\text{num-partitions} * \text{num-Local-segments})$$

- Set the scaling factor high enough so rebalance can transfer local segments to satisfy the skew threshold, but small enough so the number of storage containers does not result in too many ROS containers, and cause ROS *pushback*. The maximum number of ROS containers is 1024.

Use the [SET_SCALING_FACTOR](#) function to change your database's scaling factor. The scaling factor can be an integer between 1 and 32.

```
=> SELECT SET_SCALING_FACTOR(12);  
SET_SCALING_FACTOR  
-----  
SET  
(1 row)
```

Local Data Segmentation

By default, the scaling factor only has an effect when Vertica rebalances the database. During rebalancing, nodes break the projection segments they contain into storage containers which they can quickly move to other nodes.

This process is more efficient than re-segmenting the entire projection (in particular, less free disk space is required), but it still has significant overhead, since storage containers have to be separated into local segments, some of which are then transferred to other nodes. This overhead is not a problem if you rarely add or remove nodes from your database.

However, if your database is growing rapidly and is constantly busy, you may find the process of adding nodes becomes disruptive. In this case, you can enable local segmentation, which tells Vertica to always segment its data based on the scaling factor, so the data is always broken into containers that are easily moved. Having the data segmented in this way dramatically speeds up the process of adding or removing nodes, since the data is always in a state that can be quickly relocated to another node. The rebalancing process that Vertica

performs after adding or removing a node just has to decide which storage containers to relocate, instead of first having to first break the data into storage containers.

Local data segmentation increases the number of storage containers stored on each node. This is not an issue unless a table contains many partitions. For example, if the table is partitioned by day and contains one or more years. If local data segmentation is enabled, then each of these table partitions is broken into multiple local storage segments, which potentially results in a huge number of files which can lead to ROS "pushback." Consider your table partitions and the effect enabling local data segmentation may have before enabling the feature.

Enabling and Disabling Local Segmentation

To enable local segmentation, use the [ENABLE_LOCAL_SEGMENTS](#) function. To disable local segmentation, use the [DISABLE_LOCAL_SEGMENTATION](#) function:

```
=> SELECT ENABLE_LOCAL_SEGMENTS();
ENABLE_LOCAL_SEGMENTS
-----
ENABLED
(1 row)

=> SELECT is_local_segment_enabled FROM elastic_cluster;
is_enabled
-----
t
(1 row)

=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
-----
DISABLED
(1 row)

=> SELECT is_local_segment_enabled FROM ELASTIC_CLUSTER;
is_enabled
-----
f
(1 row)
```

Elastic Cluster Best Practices

The following are some best practices with regard to local segmentation.

Note: You should always perform a database backup before and after performing any of the operations discussed in this topic. You need to back up before changing any elastic cluster or local segmentation settings to guard against a hardware failure causing the rebalance process to leave the database in an unusable state. You should perform a full backup of the database after the rebalance procedure to avoid having to rebalance the

database again if you need to restore from a backup.

When to Enable Local Data Segmentation

[Local Data Segmentation](#) can significantly speed up the process of resizing your cluster. You should enable local data segmentation if:

- your database does not contain tables with hundreds of partitions.
- the number of nodes in the database cluster is a power of two.
- you plan to expand or contract the size of your cluster.

Local segmentation can result in an excessive number of storage containers with tables that have hundreds of partitions, or in clusters with a non-power-of-two number of nodes. If your database has these two features, take care when enabling local segmentation.

Monitoring Elastic Cluster Rebalancing

Vertica includes system tables that can be used to monitor the rebalance status of an elastic cluster and gain general insight to the status of elastic cluster on your nodes.

- The [REBALANCE_TABLE_STATUS](#) table provides general information about a rebalance. It shows, for each table, the amount of data that has been separated, the amount that is currently being separated, and the amount to be separated. It also shows the amount of data transferred, the amount that is currently being transferred, and the remaining amount to be transferred (or an estimate if storage is not separated).

Note: If multiple rebalance methods were used for a single table (for example, the table has unsegmented and segmented projections), the table may appear multiple times - once for each rebalance method.

- [REBALANCE_PROJECTION_STATUS](#) can be used to gain more insight into the details for a particular projection that is being rebalanced. It provides the same type of information as above, but in terms of a projection instead of a table.

In each table, *separated_percent* and *transferred_percent* can be used to determine overall progress.

Historical Rebalance Information

Historical information about work completed is retained, so use the predicate "*where is_latest*" to restrict the output to only the most recent or current rebalance activity. The historical data may include information about dropped projections or tables. If a table or projection has been dropped and information about the anchor table is not available, then NULL is displayed for the `table_id` and "<unknown>" is displayed for the `table_name`. Information on dropped tables is still useful, for example, in providing justification for the duration of a task.

Adding Nodes

There are many reasons for adding one or more nodes to an installation of Vertica:

- **Increase system performance.** Add additional nodes due to a high query load or load latency or increase disk space without adding storage locations to existing nodes.

Note: The database response time depends on factors such as type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if the response time is already short, e.g., less than 10 seconds, or the response time is not hardware bound.

- **Make the database K-safe** (K-safety=1) or increase K-safety to 2. See [Failure Recovery](#) for details.
- **Swap a node for maintenance.** Use a spare machine to temporarily take over the activities of an existing node that needs maintenance. The node that requires maintenance is known ahead of time so that when it is temporarily removed from service, the cluster is not vulnerable to additional node failures.
- **Replace a node.** Permanently add a node to remove obsolete or malfunctioning hardware.

Important: : If you installed Vertica on a single node without specifying the IP address or hostname (or you used localhost), you cannot expand the cluster. You must reinstall Vertica and specify an IP address or hostname that is not localhost/127.0.0.1.

Adding nodes consists of the following general tasks:

1. [Back up the database.](#)

Micro Focus strongly recommends that you back up the database before you perform this significant operation because it entails creating new projections, refreshing them, and then deleting the old projections. See [Backing Up and Restoring the Database](#) for more information.

The process of migrating the projection design to include the additional nodes could take a while; however during this time, all user activity on the database can proceed normally, using the old projections.

2. Configure the hosts you want to add to the cluster.

See [Before you Install Vertica](#) in Installing Vertica. You will also need to edit the hosts configuration file on all of the existing nodes in the cluster to ensure they can resolve the new host.

3. [Add one or more hosts to the cluster.](#)
4. [Add the hosts](#) you added to the cluster (in step 3) to the database.

Note: When you add a "host" to the database, it becomes a "node." You can add nodes to your database using either the Administration Tools or the Management Console (See [Monitoring Vertica Using Management Console.](#))

You can also add nodes using the `admintools` command line, which allows you to preserve the specific order of the nodes you add.

After you add one or more nodes to the database, Vertica automatically distributes updated configuration files to the rest of the nodes in the cluster and starts the process of rebalancing data in the cluster. See [Rebalancing Data Across Nodes](#) for details.

Adding Hosts to a Cluster

After you have backed up the database and configured the hosts you want to add to the cluster, you can now add hosts to the cluster using the `update_vertica` script.

You can use MC to add standby nodes to a database, but you cannot add hosts to a cluster using MC.

Prerequisites and Restrictions

- If you installed Vertica on a single node without specifying the IP address or hostname (you used `localhost`), it is not possible to expand the cluster. You must reinstall Vertica and specify an IP address or hostname.
- If your database has more than one node already, you can add a node without stopping the server. However, if you are adding a node to a single-node installation, then you must shut down both the database and spread. If you do not, the system returns an error like the following:

```
$ sudo /opt/vertica/sbin/update_vertica --add-hosts v_vmart_node0005 --rpm vertica_8.1.x.x86_64.RHEL6.rpm
Vertica 7.0.x Installation Tool
Starting installation tasks...
Getting system information for cluster (this may take a while)...
Spread is running on ['v_vmart_node0001']. Vertica and spread must be stopped before adding nodes to a 1 node cluster.
Use the admin tools to stop the database, if running, then use the following command to stop spread:
    /etc/init.d/spread stop (as root or with sudo)
Installation completed with errors.
Installation failed.
```

Procedure to Add Hosts

From one of the existing cluster hosts, run the `update_vertica` script with a minimum of the `--add-hosts host(s)` parameter (where *host(s)* is the hostname or IP address of the system (s) that you are adding to the cluster) and the `--rpm` or `--deb` parameter:

```
# /opt/vertica/sbin/update_vertica --add-hosts host(s) --rpm package
```

Note: See [Installing Vertica with the Installation Script](#) for the full list of parameters. You must also provide the same options you used when originally installing the cluster.

The `update_vertica` script uses all the same options as `install_vertica` and:

- Installs the Vertica RPM on the new host.
- Performs post-installation checks, including RPM version and N-way network connectivity checks.
- Modifies spread to encompass the larger cluster.
- Configures the [Administration Tools](#) to work with the larger cluster.

Important Tips:

- Consider using `--large-cluster` with more than 50 nodes.
- A host can be specified by the hostname or IP address of the system you are adding to the cluster. However, internally Vertica stores all host addresses as IP addresses.
- Do not use include spaces in the hostname/IP address list provided with `--add-hosts` if you specified more than one host.

- If a package is specified with `--rpm/--deb`, and that package is newer than the one currently installed on the existing cluster, then, Vertica first installs the new package on the existing cluster hosts before the newly-added hosts.
- Use the same command line parameters for the database administrator username, password, and directory path you used when you installed the cluster originally. Alternatively, you can create a properties file to save the parameters during install and then re-using it on subsequent install and update operations. See [Installing Vertica Silently](#).
- If you are installing using `sudo`, the database administrator user (`dbadmin`) must already exist on the hosts you are adding and must be configured with passwords and home directory paths identical to the existing hosts. Vertica sets up passwordless ssh from existing hosts to the new hosts, if needed.
- If you initially used the `--point-to-point` option to configure spread to use direct, point-to-point communication between nodes on the subnet, then use the `--point-to-point` option whenever you run `install_vertica` or `update_vertica`. Otherwise, your cluster's configuration is reverted to the default (*broadcast*), which may impact future databases.
- The maximum number of spread daemons supported in point-to-point communication and broadcast traffic is 80.
 - It is possible to have more than 80 nodes by using large cluster mode, which does not install a spread daemon on each node.

Examples:

```
--add-hosts host01 --rpm
--add-hosts 192.168.233.101
--add-hosts host02,host03
```

Adding Nodes to a Database

Once you have added one or more hosts to the cluster, you can add them as nodes to the database.

You can add nodes to a database using these methods:

- The Management Console interface
- The Administration Tools interface

- The `admintools` command line (to preserve the specific order of the nodes you add)

Note: When you add nodes to a database using either the MC or the Administration Tools interface (GUI), Vertica does not always preserve the order of the host and node names. You can preserve the order of the nodes you add by using the `admintools` command line with the `db_add_node` tool. An example follows, where the `-s` option allows you to specify the order of the nodes you are adding.

```
$ admintools -t db_add_node -d sampleDB -p 'password' -s  
192.0.2.1,192.0.2.2,192.0.2.3
```

To Add Nodes to a Database Using MC

Only nodes in STANDBY state are eligible for addition. STANDBY nodes are nodes included in the cluster but not yet assigned to the database.

You add nodes to a database on MC's **Manage** page. Click the node you want to act upon, and then click **Add node** in the Node List.

When you add a node, the node icon in the cluster view changes color from gray (empty) to green as the node comes online. Additionally, a task list displays detailed progress of the node addition process.

To Add Nodes to a Database Using the Administration Tools:

1. Open the Administration Tools. (See [Using the Administration Tools.](#))
2. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it is not, start it.
3. From the **Main Menu**, select **Advanced Tools Menu** and click **OK**.
4. In the **Advanced Menu**, select **Cluster Management** and click **OK**.
5. In the **Cluster Management** menu, select **Add Host(s)** and click **OK**.
6. Select the database to which you want to add one or more hosts, and then select **OK**.
A list of unused hosts is displayed.
7. Select the hosts you want to add to the database and click **OK**.
8. When prompted, click **Yes** to confirm that you want to add the hosts.

9. When prompted, enter the password for the database, and then select **OK**.
10. When prompted that the hosts were successfully added, select **OK**.
11. Vertica now automatically starts the rebalancing process to populate the new node with data. When prompted, enter the path to a temporary directory that the Database Designer can use to rebalance the data in the database and select **OK**.
12. Either press enter to accept the default K-Safety value, or enter a new higher value for the database and select **OK**.
13. Select whether Vertica should immediately start [rebalancing the database](#), or whether it should create a script to rebalance the database later. You should select the option to automatically start rebalancing unless you want to delay rebalancing until a time when the database has a lower load. If you choose to automatically rebalance the database, the script is still created and saved where you can use it later.
14. Review the summary of the rebalancing process and select **Proceed**.
15. If you chose to automatically rebalance, the rebalance process runs. If you chose to create a script, the script is generated and saved. In either case, you are shown a success screen, and prompted to select **OK** to end the Add Node process.

Removing Nodes

Although less common than adding a node, permanently removing a node is useful if the host system is obsolete or over-provisioned.

Note: You cannot remove nodes if doing so leaves your cluster without the minimum number of nodes required to maintain your database's current K-safety level (3 nodes for a database with a K-safety level of 1, and 5 nodes for a K-safety level of 2). If you really wish to remove the node or nodes from the database, you first must reduce the K-safety level of your database.

This section includes:

Automatic Eviction of Unhealthy Nodes

To decrease the impact of an unhealthy node in your Vertica database, Vertica performs regular health checks. The health checks are performed on a regular schedule. The time between each interval is set by the user using the DatabaseHeartBeatInterval parameter. This parameter specifies the time intervals between internal health checks performed by each node. After a successful health check, the node sends a heartbeat. If a heartbeat is not detected by the time five intervals have elapsed, then the node is evicted from the database cluster.

The formula for the amount of time allowed before an eviction is

$$TOT = DHBI * 5$$

where *TOT* is the total time (in seconds) allowed without a heartbeat before eviction, and *DHBI* is equal to the value of DatabaseHeartBeatInterval.

By default DatabaseHeartBeatInterval is set to 120, which allows five 120-second intervals to pass without a heartbeat. If you set the DatabaseHeartBeatInterval too low, it can cause evictions in cases of brief node health issues. Sometimes, such premature evictions result in lower availability and performance of the Vertica database.

See Also

DatabaseHeartbeatInterval in [General Parameters](#)

Lowering K-Safety to Enable Node Removal

A database with a K-safety level of 1 requires at least three nodes to operate, and a database with a K-Safety level 2 requires at least 5 nodes to operate. To remove a node from a cluster that is at the minimum number of nodes for its database's K-safety level, first lower the K-safety level with [MARK_DESIGN_KSAFE](#).

Caution: Lowering the K-safety level of a database to 0 eliminates Vertica's fault tolerance features. If you must reduce K-safety to 0, first [back up the database](#).

To lower the K-safety level of the database:

1. Connect to the database with [Administration Tools](#) or `vsq`.
2. Call the function `MARK_DESIGN_KSAFE`:

```
SELECT MARK_DESIGN_KSAFE(n);
```

where *n* is the new K-safety level for the database.

Removing Nodes From a Database

As long as there are enough nodes remaining to satisfy the K-Safety requirements, you can remove the node from a database. You cannot drop nodes that are critical for K-safety. See [Lowering K-Safety to Enable Node Removal](#)

You can remove nodes from a database using one of the following methods:

- The Management Console interface
- The Administration Tools interface

Prerequisites

Before attempting to remove a node from the database, ensure the following prerequisites are met:

- Your database must be running.
- You must [back up the database](#).

- If necessary, [lower the K-safety of your database](#) if the cluster will not be large enough to support its current level of K-safety after you remove nodes. Remember you must have enough nodes to satisfy K-Safety requirements.

Remove Hosts From the Database with Management Console

Remove nodes with Management Console from the **Manage** page. You can only remove nodes that belong to the database cluster and have a DOWN state (red). Remove nodes as follows:

1. Select the node you want to remove.
2. Click **Remove node** in the Node List.

When you remove a node the state changes to STANDBY. You can add STANDBY nodes back to the database later, see [Using the Management Console to Replace Nodes](#).

Remove Hosts From the Database with Administration Tools

To remove unused hosts from the database using Administration Tools:

1. Open the Administration Tools. See [Using the Administration Tools](#) for information about accessing the Administration Tools.
2. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If the database is not running, start it.
3. From the **Main Menu**, select **Advanced Tools Menu** and select **OK**.
4. In the **Advanced** menu, select **Cluster Management** and select **OK**.
5. In the **Cluster Management** menu, select **Remove Host(s) from Database** and select **OK**.
6. When warned that you must redesign your database and create projections that exclude the hosts you are going to drop, select **Yes**.
7. Select the database from which you want to remove the hosts and select **OK**.

A list of currently active hosts appears.

8. Select the hosts you want to remove from the database and select **OK**.
9. When prompted, select **OK** to confirm that you want to remove the hosts.
10. When informed that the hosts were successfully removed, select **OK**.

11. If you removed a host from a [Large Cluster](#) configuration, open a vsql session and run the following command:

```
SELECT realign_control_nodes();
```

For more details, see [REALIGN_CONTROL_NODES](#).

12. If this host is not used by any other database in the cluster, you can remove the host from the cluster. See [Removing Hosts From a Cluster](#).

Removing Hosts From a Cluster

If a host that you removed from the database is not used by any other database, you can remove it from the cluster using the `update_vertica` script. You can leave the database running (UP) during this operation.

You can [remove hosts from a database](#) on the MC interface, but you cannot remove those hosts from a cluster.

Prerequisites

The host must not be used by any database.

Procedure to Remove Hosts

From one of the hosts in the cluster, run `update_vertica` with the `--remove-hosts` switch. Provide a comma-separated list of hosts to remove from an existing Vertica cluster. You can specify a host by the host name or IP address of the system.

This example removes `host01`, `host02`, and `host03` from the cluster:

```
# /opt/vertica/sbin/update_vertica --remove-hosts host01,host02,host03
```

Note: See [Installing Vertica with the Installation Script](#) for the full list of parameters.

The `update_vertica` script uses all the same options as `install_vertica` and:

- Modifies the spread to match the smaller cluster.
- Configures the Administration Tools to work with the smaller cluster.

Important Tips

- Do not include spaces in the hostname list provided with `--remove-hosts` if you specified more than one host.
- If a new RPM is specified with `--rpm`, then Vertica will first install it on the existing cluster hosts before proceeding.
- Use the same command line parameters as those used when you installed the original cluster. Specifically if you used non-default values for the database administrator username, password, or directory path, provide the same when you remove hosts; otherwise, the procedure fails. Consider creating a properties file in which you save the parameters during the installation, which you can reuse on subsequent install and update operations. See [Installing Vertica Silently](#).

Replacing Nodes

If you have a K-Safe database, you can replace nodes, as necessary, without bringing the system down. For example, you might want to replace an existing node if you:

- Need to repair an existing host system that no longer functions and restore it to the cluster
- Want to exchange an existing host system for another more powerful system

Note: Vertica does not support replacing a node on a K-safe=0 database. Use the procedures to [add](#) and [remove](#) nodes instead.

The process you use to replace a node depends on whether you are replacing the node with:

- A host that uses the same name and IP address
- A host that uses a different name and IP address
- An active standby node

Prerequisites

- Configure the replacement hosts for Vertica. See [Before you Install Vertica](#) in [Installing Vertica](#).
- Read the Important **Tips** sections under [Adding Hosts to a Cluster](#) and [Removing Hosts From a Cluster](#).
- Ensure that the database administrator user exists on the new host and is configured identically to the existing hosts. Vertica will setup passwordless ssh as needed.
- Ensure that directories for Catalog Path, Data Path, and any storage locations are added to the database when you create it and/or are mounted correctly on the new host and have read and write access permissions for the database administrator user. Also ensure that there is sufficient disk space.
- Follow the best practice procedure below for introducing the failed hardware back into the cluster to avoid spurious full-node rebuilds.

Best Practice for Restoring Failed Hardware

Following this procedure will prevent Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

1. Reboot the machine into runlevel 1, which is a root and console-only mode.

Runlevel 1 prevents network connectivity and keeps Vertica from attempting to reconnect to the cluster.

2. In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

3. Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

At this point, the network activates, and Vertica rejoins the cluster and automatically recovers any missing data. Note that, on a single-node database, if any files that were associated with a projection have been deleted or corrupted, Vertica will delete all files associated with that projection, which could result in data loss.

Replacing a Host Using the Same Name and IP Address

If a host of an existing Vertica database is removed you can replace it while the database is running.

Note: Remember a host in Vertica consists of the hardware and operating system on which Vertica software resides, as well as the same network configurations.

You can replace the host with a new host that has the following same characteristics as the old host:

- Name
- IP address

- Operating system
- The OS administrator user
- Directory location

Replacing the host while your database is running prevents system downtime. Before replacing a host, backup your database. See [Backing Up and Restoring the Database](#) for more information.

Replace a host using the same characteristics as follows:

1. Run `install_vertica` from a functioning host using the `--rpm` or `--deb` parameter:

```
$ /opt/vertica/sbin/install_vertica --rpm <rpm_package>
```

For more information see [Installing Vertica](#).

2. Use Administration Tools from an existing node to restart the new host. See [Restart Vertica on a Node](#).

The node automatically joins the database and recovers its data by querying the other nodes in the database. It then transitions to an UP state.

Replacing a Failed Node Using a Node with a Different IP Address

Replacing a failed node with a host system that has a different IP address from the original consists of the following steps:

1. [Back up the database](#).

Micro Focus recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. Add the new host to the cluster. See [Adding Hosts to a Cluster](#).
3. If Vertica is still running in the node being replaced, then use the Administration Tools to *Stop Vertica on Host* on the host being replaced.

4. Use the Administration Tools to [replace the original host](#) with the new host. If you are using more than one database, replace the original host in all the databases in which it is used. See [Replacing Hosts](#).
5. Use the procedure in [Distributing Configuration Files to the New Host](#) to transfer metadata to the new host.
6. [Remove the host from the cluster](#).
7. Use the Administration Tools to restart Vertica on the host. On the **Main Menu**, select **Restart Vertica on Host**, and click **OK**. See [Start the Database](#) for more information.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying other nodes within the database.

Replacing a Functioning Node Using a Different Name and IP Address

Replacing a node with a host system that has a different IP address and host name from the original consists of the following general steps:

1. [Back up the database](#).

Micro Focus recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. [Add the replacement hosts to the cluster](#).

At this point, both the original host that you want to remove and the new replacement host are members of the cluster.

3. Use the Administration Tools to Stop *Vertica on Host* on the host being replaced.
4. Use the Administration Tools to [replace the original host](#) with the new host. If you are using more than one database, replace the original host in all the databases in which it is used. See [Replacing Hosts](#).
5. [Remove the host from the cluster](#).
6. Restart Vertica on the host.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying the other nodes within the database. It then transitions to an UP state.

Note: If you do not remove the original host from the cluster and you attempt to restart the database, the host is not invited to join the database because its node address does not match the new address stored in the database catalog. Therefore, it remains in the INITIALIZING state.

Using the Administration Tools to Replace Nodes

If you are replacing a node with a host that uses a different name and IP address, use the Administration Tools to replace the original host with the new host. Alternatively, you can [use the Management Console to replace a node](#).

Replace the Original Host with a New Host Using the Administration Tools

To replace the original host with a new host using the Administration Tools:

1. Back up the database. See [Backing Up and Restoring the Database](#).
2. From a node that is up, and is not going to be replaced, open the Administration Tools.
3. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it's not running, use the Start Database command on the Main Menu to restart it.
4. On the **Main Menu**, select **Advanced Menu**.
5. In the **Advanced Menu**, select **Stop Vertica on Host**.
6. Select the host you want to replace, and then click **OK** to stop the node.
7. When prompted if you want to stop the host, select **Yes**.
8. In the **Advanced Menu**, select **Cluster Management**, and then click **OK**.
9. In the **Cluster Management** menu, select **Replace Host**, and then click **OK**.
10. Select the database that contains the host you want to replace, and then click **OK**.

A list of all the hosts that are currently being used displays.

11. Select the host you want to replace, and then click **OK**.
12. Select the host you want to use as the replacement, and then click **OK**.
13. When prompted, enter the password for the database, and then click **OK**.
14. When prompted, click **Yes** to confirm that you want to replace the host.
15. When prompted that the host was successfully replaced, click **OK**.
16. In the **Main Menu**, select **View Database Cluster State** to verify that all the hosts are running. You might need to start Vertica on the host you just replaced. Use **Restart Vertica on Host**.

The node enters a RECOVERING state.

Caution: If you are using a K-Safe database, keep in mind that the recovering node counts as one node down even though it might not yet contain a complete copy of the data. This means that if you have a database in which $K \text{ safety}=1$, the current fault tolerance for your database is at a critical level. If you lose one more node, the database shuts down. Be sure that you do not stop any other nodes.

Using the Management Console to Replace Nodes

On the MC **Manage** page, you can quickly replace a DOWNS node in the database by selecting one of the STANDBY nodes in the cluster.

A DOWNS node shows up as a red node in the cluster. Click the DOWNS node and the Replace node button in the Node List becomes activated, as long as there is at least one node in the cluster that is not participating in the database. The STANDBY node will be your replacement node for the node you want to retire; it will appear gray (empty) until it has been added to the database, when it turns green.

Tip: You can resize the Node List by clicking its margins and dragging to the size you want.

When you highlight a node and click **Replace**, MC provides a list of possible STANDBY nodes to use as a replacement. After you select the replacement node, the process begins. A node replacement could be a long-running task.

MC transitions the DOWNS node to a STANDBY state, while the node you selected as the replacement will assume the identity of the original node, using the same node name, and will be started.

Assuming a successful startup, the new node will appear orange with a status of RECOVERING until the recovery procedure is complete. When the recovery process completes, the replacement node will turn green and show a state of UP.

Rebalancing Data Across Nodes

Vertica can rebalance your database when you add or remove nodes. As a superuser, you can manually trigger a rebalance with [Administration Tools](#), [SQL functions](#), or the [Management Console](#).

A rebalance operation can take some time, depending on the cluster size, and the number of projections and the amount of data they contain. You should allow the process to complete uninterrupted. If you must cancel the operation, call [CANCEL_REBALANCE_CLUSTER](#).

Why Rebalance?

Rebalancing is useful or even necessary after you perform one of the following operations:

- Change the size of the cluster by adding or removing nodes.
- Mark one or more nodes as ephemeral in preparation of removing them from the cluster.
- Change the [scaling factor](#) of an elastic cluster, which determines the number of storage containers used to store a projection across the database.
- Set the control node size or realign control nodes on a [large cluster](#) layout.
- Specify more than 120 nodes in your initial Vertica cluster configuration.
- Modify a [fault group](#) by adding or removing nodes.

General Rebalancing Tasks

When you rebalance a database cluster, Vertica performs the following tasks for all projections, segmented and unsegmented alike:

- Distributes data based on:
 - User-defined [fault groups](#), if specified
 - [Large cluster](#) automatic fault groups
- Ignores node-specific distribution specifications in projection definitions. Node rebalancing always distributes data across all nodes.

- When rebalancing is complete, sets the Ancient History Mark the greatest allowable epoch (now).

Vertica rebalances segmented and unsegmented projections differently, as described below.

Rebalancing Segmented Projections

For each segmented projection, Vertica performs the following tasks:

1. Copies and renames projection buddies and distributes them evenly across all nodes. The renamed projections share the same base name.
2. Refreshes the new projections.
3. Drops the original projections.

Rebalancing Unsegmented Projections

For each unsegmented projection, Vertica performs the following tasks:

If adding nodes:

- Creates projection buddies on them.
- Maps the new projections to their shared name in the database catalog.

If dropping nodes: drops the projection buddies from them.

K-safety and Rebalancing

Until rebalancing completes, Vertica operates with the existing K-safe value. After rebalancing completes, Vertica operates with the K-safe value specified during the rebalance operation.

The new K-safe value must be equal to or higher than current K-safety. Vertica does not support downgrading K-safety and returns a warning if you try to reduce it from its current value. For more information, see [Lowering K-Safety to Enable Node Removal](#).

Rebalancing Failure and Projections

If a failure occurs while rebalancing the database, you can rebalance again. If the cause of the failure has been resolved, the rebalance operation continues from where it failed. However, a

failed data rebalance can result in projections becoming out of date.

To locate any such projections, query the system table [PROJECTIONS](#) as follows:

```
=> SELECT projection_name, anchor_table_name, is_prejoin, is_up_to_date FROM projections  
WHERE is_up_to_date = false;
```

To remove out-of-date projections, use [DROP PROJECTION](#).

Temporary Tables

Node rebalancing has no effect on projections of temporary tables.

Rebalancing Data Using the Administration Tools UI

To rebalance the data in your database:

1. Open the Administration Tools. (See [Using the Administration Tools](#).)
2. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it is not, start it.
3. From the **Main Menu**, select **Advanced Menu** and click **OK**.
4. In the **Advanced Menu**, select **Cluster Management** and click **OK**.
5. In the **Cluster Management** menu, select **Re-balance Data** and click **OK**.
6. Select the database you want to rebalance, and then select **OK**.
7. Enter the directory for the Database Designer outputs (for example `/tmp`) and click **OK**.
8. Accept the proposed K-safety value or provide a new value. Valid values are 0 to 2.
9. Review the message and click **Proceed** to begin rebalancing data.

The Database Designer modifies existing projections to rebalance data across all database nodes with the K-safety you provided. A script to rebalance data, which you can run manually at a later time, is also generated and resides in the path you specified; for example `/tmp/extend_catalog_rebalance.sql`.

Important: Rebalancing data can take some time, depending on the number of projections and the amount of data they contain. Micro Focus recommends that you allow the process to complete. If you must cancel the operation, use Ctrl+C.

The terminal window notifies you when the rebalancing operation is complete.

10. Press **Enter** to return to the Administration Tools.

Rebalancing Data Using Management Console

Vertica can rebalance the database when you add or remove nodes. If you notice data skew where one node shows more activity than another (for example, most queries processing data on a single node), you can manually rebalance the database using MC if that database is imported into the MC interface.

On the **Manage** page, click **Rebalance** in the toolbar to initiate the rebalance operation.

During a rebalance, you cannot perform any other activities on the database cluster, such as start, stop, add, or remove nodes.

Rebalancing Data Using SQL Functions

Vertica has three SQL functions for starting and stopping a cluster rebalance. You can call these functions from a script that runs during off-peak hours, rather than manually trigger a rebalance through Administration Tools.

- [REBALANCE_CLUSTER](#) rebalances the database cluster synchronously as a session foreground task.
- [START_REBALANCE_CLUSTER](#) asynchronously rebalances the database cluster as a background task.
- [CANCEL_REBALANCE_CLUSTER](#) stops any rebalance task that is currently in progress or is waiting to execute.

Redistributing Configuration Files to Nodes

The add and remove node processes automatically redistribute the Vertica configuration files. You rarely need to redistribute the configuration files to help resolve configuration issues.

To distribute configuration files to a host:

1. Log on to a host that contains these files and start the Administration Tools.

See [Using the Administration Tools](#) for information about accessing the Administration Tools.

2. On the **Main Menu** in the Administration Tools, select **Configuration Menu** and click **OK**.
3. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
4. Select **Database Configuration**.
5. Select the database where you want to distribute the files and click **OK**.

The `vertica.conf` file is distributed to all the other hosts in the database. If it previously existed on a host, it is overwritten.

6. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
7. Select **SSL Keys**.

The certifications and keys for the host are distributed to all the other hosts in the database. If they previously existed on a host, they are overwritten.

8. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.

Select **AdminTools Meta-Data**.

The Administration Tools metadata is distributed to every host in the cluster.

9. [Restart the database](#).

Note: To distribute configuration files from the command line or via scripts, use the `admintools` option `distribute_config_files`:

```
admintools -t distribute_config_files
```

Stopping and Starting Nodes on MC

You can start and stop one or more database nodes through the **Manage** page by clicking a specific node to select it and then clicking the Start or Stop button in the Node List.

Note: The Stop and Start buttons in the toolbar start and stop the database, not individual nodes.

On the **Databases and Clusters** page, you must click a database first to select it. To stop or start a node on that database, click the **View** button. You'll be directed to the Overview page. Click **Manage** in the applet panel at the bottom of the page and you'll be directed to the database node view.

The Start and Stop database buttons are always active, but the node Start and Stop buttons are active only when one or more nodes of the same status are selected; for example, all nodes are UP or DOWN.

After you click a Start or Stop button, Management Console updates the status and message icons for the nodes or databases you are starting or stopping.

Mapping New IP Addresses

There are times when existing, operational Vertica database cluster nodes need to run on different IP addresses. Cluster nodes may also need to run based on different IP protocols, for example, when changing the protocol from broadcast to point-to-point. To run Vertica in these situations, use the `re_ip` function to re-IP and map the old addresses to the new addresses:

```
$ admintools -t re_ip -f mapfile
```

The *mapfile* references the file that you must create. A map file contains the old and new IP addresses. For details see [Re-IP Addresses with a Mapping File](#).

Use this function to re-IP in one of the following situations:

- If your Vertica database cluster has the same data and control messaging address, do one of the following:
 - Re-IP all the database cluster node IP addresses.
 - Re-IP only one or some of the database cluster node IP addresses.

```
$ admintools -t re_ip -f mapfile
```

- Re-IP the Vertica database cluster from broadcast mode to point-to-point (unicast) mode:

```
$ admintools -t re_ip -d db name -T
```

- Re-IP the Vertica database cluster from point-to-point (unicast) mode to broadcast mode :

```
$ admintools -t re_ip -d db name -U
```

Note: For information on changing communications protocols, see the -U and -T options under [install_vertica Options](#)

- Re-IP the control address of the database cluster. In this case the mapping file must contain the control messaging IP address and associated broadcast address.

```
$ admintools -t re_ip -f mapfile
```

- Re-IP only one database address without changing the admintools configuration. See [Mapping IP Addresses on the Database only](#).

Note: The database only re-ip is useful for error recovery. The node names and IP addresses must be the same as the node information in admintools.conf. You can also run `SELECT * from vs_nodes order by name` to display the node information.

For more information on the options used in the above commands, see [Re-IP Command-Line Options](#).

Re-IP and the Export IP address

By default, the node IP address and the export IP address are configured with the same IP address. The export address is the IP address of the node on the network with access to other DBMS systems. Use the export address for importing and exporting data from DBMS systems. You can manually change the export address using the instructions found [here](#).

If you change the export address and run the re-ip command, the export address remains the same.

Example

Run the following command:

```
=> SELECT node_name, node_address, export_address FROM nodes;
   node_name      | node_address  | export_address
-----|-----|-----
v_VMartDB_node0001 | 192.168.100.101 | 192.168.100.101
v_VMartDB_node0002 | 192.168.100.102 | 192.168.100.101
v_VMartDB_node0003 | 192.168.100.103 | 192.168.100.101
v_VMartDB_node0004 | 192.168.100.104 | 192.168.100.101
(4 rows)
```

In the above example the `export_address` is the default. In this case, when you run the re-ip command the `export_address` changes to the new `node_address`.

If you manually change the export address as [described here](#), you may have something like the following:

```
=> SELECT node_name, node_address, export_address FROM nodes;
   node_name      | node_address      | export_address
-----|-----|-----
v_VMartDB_node0001 | 192.168.100.101 | 10.10.10.1
v_VMartDB_node0002 | 192.168.100.102 | 10.10.10.2
v_VMartDB_node0003 | 192.168.100.103 | 10.10.10.3
v_VMartDB_node0004 | 192.168.100.104 | 10.10.10.4
(4 rows)
```

In this case, when you run the re-ip command the export_address does not change.

Finding IP Addresses

IP addresses for the hosts and nodes are stored in `opt/vertica/config/admintools.conf`:

```
[Cluster]
hosts = 203.0.113.111, 203.0.113.112, 203.0.113.113

[Nodes]
node0001 = 203.0.113.111/home/dbadmin,/home/dbadmin
node0002 = 203.0.113.112/home/dbadmin,/home/dbadmin
node0003 = 203.0.113.113/home/dbadmin,/home/dbadmin
```

You can also display a list of IP addresses with the following:

```
$ admintools -t list_allnodes
Node          | Host          | State | Version          | DB
-----|-----|-----|-----|-----
v_vmart_node0001 | 203.0.113.111 | UP    | vertica-8.1.1.20170511 | VMart
v_vmart_node0002 | 203.0.113.112 | UP    | vertica-8.1.1.20170511 | VMart
v_vmart_node0003 | 203.0.113.113 | UP    | vertica-8.1.1.20170511 | VMart
```

Tip: Run the `list_allnodes` tool to help identify any issues you may be having accessing Vertica. For example, if the hosts are not communicating with each other, `Unavailable` appears in the `Version` column.

Re-IP Addresses with a Mapping File

Mapping new IP addresses includes:

- Creating a mapping file that maps the old IP addresses to the new IP addresses.
- Using the mapping file to update configuration files and the database catalog.

Note: If you are using control messaging for communication between hosts, you may also need to update IPs with the new controlAddress and controlBroadcast IP addresses.

The embedded messages subsystem operates based on the controlAddress IP and controlBroadcast IP when you use the -U option.

Create Mapping File

Before creating a mapping file you need to know the new IP addresses. Create a mapping file as follows:

1. If you do not already have them, obtain the new IP addresses and save them in a text file. You can save the file anywhere on your system.
2. Run the following command to obtain the old IP addresses.

```
$ admintools -t list_allnodes
Node          | Host          | State | Version          | DB
-----+-----+-----+-----+-----
v_vmart_node0001 | 192.0.2.254 | UP    | vertica-8.1.1.20170511 | VMart
v_vmart_node0002 | 192.0.2.255 | UP    | vertica-8.1.1.20170511 | VMart
v_vmart_node0003 | 192.0.2.256 | UP    | vertica-8.1.1.20170511 | VMart
```

3. Copy the contents of the Host column into the same text file as the new IP addresses. The file is in the format old address, new address:

```
192.0.2.254 198.51.100.255
192.0.2.255 198.51.100.256
192.0.2.256 198.51.100.257
```

You can have the following map file formats:

Re-IP from an old IP address to a new IP address:

```
oldIPAddress newIPAddress, controlAddress (optional), controlBroadcast (optional)
```

In this scenario, the controlAddress and controlBroadcast are optional. If you do not include them in the map file:

- The controlAddress defaults to the newIPAddress.
- The controlBroadcast defaults to the host of the newIPAddress's broadcast IP address.

For example:

```
192.0.2.254 198.51.100.255, 198.51.100.255, 203.0.113.255  
192.0.2.255 198.51.100.256, 198.51.100.256, 203.0.113.255  
192.0.2.256 198.51.100.257, 198.51.100.257, 203.0.113.255
```

The command for performing this Re-IP process is as follows:

```
$ admintools -t re_ip -f <mapfile>
```

Re-IP from an old IP address to a new IP address and change the control messaging mode

```
oldIPAddress newIPAddress, controlAddress, controlBroadcast
```

For example:

```
192.0.2.254 198.51.100.255, 203.0.113.255, 203.0.113.258  
192.0.2.255 198.51.100.256, 203.0.113.256, 203.0.113.258  
192.0.2.256 198.51.100.257, 203.0.113.257, 203.0.113.258
```

Note that the map file uses comma separators after the new IP address and controlAddress.

The command for performing this re-IP process and changing the control messaging mode to point-to-point is as follows:

```
$ admintools -t re_ip -d db name -T
```

The command for performing this re-IP process and changing the control messaging mode to broadcast is as follows:

```
$ admintools -t re_ip -d db name -U
```

Re-IP the node control address on the database only (see [Mapping IP Addresses on the Database only](#)).

```
nodeName nodeIPAddress controlAddress, controlBroadcast
```

For example:

```
v_vmart_node0001 192.0.2.254, 203.0.113.255, 203.0.113.258  
v_vmart_node0002 192.0.2.255, 203.0.113.256, 203.0.113.258
```

```
v_vmart_node0003 192.0.2.256, 203.0.113.257, 203.0.113.258
```

The command for performing database-only re-IP is as follows:

```
$ admintools -t re_ip -f <mapfile> -O -d database
```

Re-IP the IP Addresses

After creating the mapping file you can re-IP the new IP addresses. The re-IP process automatically backs up `admintools.conf` so you can recover the original settings if necessary.

1. Stop the database.
2. Run the following command to map the old IP addresses to the new IP addresses:

```
$ admintools -t re_ip -f mapfile
```

Note: This example uses the command for performing a re-IP from an old IP address to a new IP address.

A warning occurs if:

- Any of the IP addresses is incorrectly formatted
- a duplicate old or new IP address exists in the file. For example, `192.0.2.256` appears twice in the old IP set.

If the syntax is correct and mapping begins:

- Re-maps the IP addresses as listed in the mapping file.
- Prompts you to confirm the updates to the database, unless you use the `-i` option.
- Updates the required local configuration files with the new IP addresses.
- Distributes the updated configuration files to the hosts using the new IP addresses.

Track these steps using the following prompts:

```
Parsing mapfile...  
New settings for Host 192.0.2.254 are:  
  
address: 198.51.100.255
```

```
New settings for Host 192.0.2.255 are:
address: 198.51.100.256

New settings for Host 192.0.2.254 are:
address: 198.51.100.257

The following databases would be affected by this tool: Vmart

Checking DB status ...
Enter "yes" to write new settings or "no" to exit > yes
Backing up local admintools.conf ...
Writing new settings to local admintools.conf ...

Writing new settings to the catalogs of database Vmart ...
The change was applied to all nodes.
Success. Change committed on a quorum of nodes.

Initiating admintools.conf distribution ...
Success. Local admintools.conf sent to all hosts in the cluster.
```

3. Restart the database.

Mapping IP Addresses on the Database only

You can map IP addresses for just the database. This task involves mapping the name of the nodes in the database to the new IP addresses. This is useful for error recovery because `admintools.conf` does not get updated. Vertica updates only `spread.conf` and the catalog with the changes.

You can also map IP addresses on the database only to set `controlAddress` and `controlBroadcast` on a single database. This task allows nodes on the same host to have a different data and `controlAddress`.

1. Stop the database.
2. Create a mapping file in the following format:

```
nodeName IPaddress, controlAddress, controlBroadcast
```

For example:

```
192.0.2.254, 203.0.113.255, 203.0.113.258 192.0.2.256, 203.0.113.257, 203.0.113.258
```

```
vertica_node001 192.0.2.254, 203.0.113.255, 203.0.113.258
vertica_node002 192.0.2.255, 203.0.113.256, 203.0.113.258
vertica_node003 192.0.2.256, 203.0.113.257, 203.0.113.258
```

3. Run the following command to map the new IP addresses:

```
$ admintools -t re_ip -f <mapfile> -O -d database
```

4. Restart the database.

Re-IP Command-Line Options

The table below lists the command-line options you can use with the `re_ip` command.

Option	Description
- h or --help	Displays the online help for <code>re_ip</code> .
-f <mapfile> or --file=<mapfile>	The name of the mapping text file. What this file contains depends on the type of re-IP you want to perform. See Mapping New IP Addresses .
-O or --dba-only	Used for error recovery. Updates and replaces data on the database cluster catalog and control messaging system. If the map text file fails, Vertica automatically recreates it when you re-run the command. The format of the map text file is: <pre>NodeName AssociatedNodeIPAddress, new ControlAddress, new ControlBroadcast</pre> <p>NodeName and AssociatedNodeIPAddress must be the same as those in <code>admintools.conf</code>.</p> <p>This option updates only one database at a time so you must use the <code>-d</code> option:</p> <pre>\$ admintools -t re_ip -f mapfile -O -d database</pre>
-i or --noprompts	Specifies that the system does not prompt for the validation of the new settings before performing the re-IP. Prompting is on by default.
-T or --point-to-point	Sets the control messaging to point-to-point (unicast) protocol. This option updates only one database at a time so you must use the <code>-d</code>

Option	Description
	<p>option. You do not need a mapping text file with this option.</p> <p>For information on setting point-to-point communication, see the -T option in install_vertica Options</p>
<p>-U or --broadcast</p>	<p>Sets the control messaging to broadcast protocol. This option updates only one database at a time so you must use the -d option. You do not need a mapping text file with this option.</p> <p>For information on setting broadcast communication, see the -U option in install_vertica Options</p>
<p>-d <database name> or --database=<database name></p>	<p>The database name. This option is required with the following re-IP options:</p> <ul style="list-style-type: none">• -O• -T• -U

Managing Disk Space

Vertica detects and reports low disk space conditions in the log file so that the issue can be addressed before serious problems occur. It also detects and reports low disk space conditions via [SNMP traps](#) if enabled.

Critical disk space issues are reported sooner than other issues. For example, running out of catalog space is fatal; therefore, Vertica reports the condition earlier than less critical conditions. To avoid database corruption when the disk space falls beyond a certain threshold, Vertica begins to reject transactions that update the catalog or data.

Caution: A low disk space report indicates one or more hosts are running low on disk space or have a failing disk. It is imperative to add more disk space (or replace a failing disk) as soon as possible.

When Vertica reports a low disk space condition, use the [DISK_RESOURCE_REJECTIONS](#) system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected.

For descriptions of all the system tables, see [Using System Tables](#).

To add disk space, see [Adding Disk Space to a Node](#). To replace a failed disk, see [Replacing Failed Disks](#).

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

System table	Description
DISK_STORAGE	Monitors the amount of disk storage used by the database on each node.
COLUMN_STORAGE	Monitors the amount of disk storage used by each column of each projection on each node.
PROJECTION_STORAGE	Monitors the amount of disk storage used by each projection on each node.

Adding Disk Space to a Node

This procedure describes how to add disk space to a node in the Vertica cluster.

Note: If you are adding disk space to multiple nodes in the cluster, then use the following procedure for each node, one node at a time.

To add disk space to a node:

1. If you must shut down the hardware to which you are adding disk space, then first shut down Vertica on the host where disk space is being added.
2. Add the new disk to the system as required by the hardware environment. Boot the hardware if it is was shut down.
3. Partition, format, and mount the new disk, as required by the hardware environment.
4. Create a data directory path on the new volume.

For example:

```
mkdir -p /myNewPath/myDB/host01_data2/
```

5. If you shut down the hardware, then restart Vertica on the host.
6. Open a database connection to Vertica and add a storage location to add the new data directory path. Specify the node in the [CREATE LOCATION](#), otherwise Vertica assumes you are creating the storage location on all nodes.

See [Creating Storage Locations](#) in this guide and the [CREATE LOCATION](#) statement in the SQL Reference Manual.

Replacing Failed Disks

If the disk on which the data or catalog directory resides fails, causing full or partial disk loss, perform the following steps:

1. Replace the disk and recreate the data or catalog directory.
2. Distribute the configuration file (`vertica.conf`) to the new host. See [Distributing Configuration Files to the New Host](#) for details.
3. Restart the Vertica on the host, as described in [Restart Vertica On Host](#).

See [Catalog and Data Files](#) for information about finding your `DATABASE_HOME_DIR`.

Catalog and Data Files

For the recovery process to complete successfully, it is essential that catalog and data files be in the proper directories.

In Vertica, the catalog is a set of files that contains information (metadata) about the objects in a database, such as the nodes, tables, constraints, and projections. The catalog files are replicated on all nodes in a cluster, while the data files are unique to each node. These files are installed by default in the following directories:

```
/DATABASE_HOME_DIR/DATABASE_NAME/v_db_nodexxxx_catalog/ /DATABASE_HOME_DIR/DATABASE_NAME/v_db_nodexxxx_catalog/
```

Note: `DATABASE_HOME_DIR` is the path, which you can see from the Administration Tools. See [Using the Administration Tools](#) in the Administrator's Guide for details on using the interface.

To view the path of your database:

1. Run the Administration Tools.

```
$ /opt/vertica/bin/admintools
```

2. From the Main Menu, select **Configuration Menu** and click **OK**.
3. Select **View Database** and click **OK**.
4. Select the database you want would like to view and click **OK** to see the database profile.

See [Understanding the Catalog Directory](#) for an explanation of the contents of the catalog directory.

Understanding the Catalog Directory

The catalog directory stores metadata and support files for your database. Some of the files within this directory can help you troubleshoot data load or other database issues. See [Catalog and Data Files](#) for instructions on locating your database's catalog directory. By default, it is located in the database directory. For example, if you created the VMart database in the database administrator's account, the path to the catalog directory is:

```
/home/dbadmin/VMart/v_vmart_nodennn_catalog
```

where `nodennn` is the name of the node you are logged into. The name of the catalog directory is unique for each node, although most of the contents of the catalog directory are identical on each node.

The following table explains the files and directories that may appear in the catalog directory.

Note: Do not change or delete any of the files in the catalog directory unless asked to do so by Vertica support.

File or Directory	Description
<code>bootstrap-catalog.log</code>	A log file generated as the Vertica server initially creates the database (in which case, the log file is only created on the node used to create the database) and whenever the database is restored from a backup.
<code>Catalog/</code>	Contains catalog information about the database, such as checkpoints.
<code>CopyErrorLogs/</code>	The default location for the COPY exceptions and rejections files generated when data in a bulk load cannot be inserted into the database. See Capturing Load Rejections and Exceptions for more information.
<code>DataCollector/</code>	Log files generated by the Data Collector.
<code>debug_log.conf</code>	Debugging information configuration file. For Micro Focus use only.
<code>Epoch.log</code>	Used during recovery to indicate the latest epoch that

File or Directory	Description
	contains a complete set of data.
ErrorReport.txt	A stack trace written by Vertica if the server process exits unexpectedly.
Libraries/	Contains user defined library files that have been loaded into the database See Developing User-Defined Extensions (UDxs) in Extending Vertica. Do not change or delete these libraries through the file system. Instead, use the CREATE LIBRARY , DROP LIBRARY , and ALTER LIBRARY statements.
Snapshots/	The location where backups are stored.
tmp/	A temporary directory used by Vertica's internal processes.
UDxLogs/	Log files written by user defined functions that run in fenced mode. See Fenced Mode in Extending Vertica for more information.
vertica.conf	The configuration file for Vertica.
vertica.log	The main log file generated by the Vertica server process.
vertica.pid	The process ID and path to the catalog directory of the Vertica server process running on this node.

Reclaiming Disk Space From Deleted Table Data

You can reclaim disk space from deleted table data in several ways:

- [Purge deleted records.](#)
- [Rebuild the table.](#)
- [Drop table partition.](#)

Managing Memory

`glibc` enhances performance by aggressively holding onto memory, even if that memory is not being used at the time by a process. The Linux kernel's Out of Memory (OOM) Killer may terminate the Vertica process if it uses too much memory.

You can avoid this by setting the `MALLOC_ARENA_MAX` environment variable to a nonzero value, which limits the number of arenas (memory pools available to `malloc()`) a process can use.

If you have write privileges on `~/ .bashrc`, perform the following steps on all nodes:

1. Run the following command to set `MALLOC_ARENA_MAX`:

```
$ export MALLOC_ARENA_MAX=4
```

2. Log out and back in.
3. Restart the Vertica database.
4. Verify that `MALLOC_ARENA_MAX` is set:

```
$ set | grep ARENA  
MALLOC_ARENA_MAX=4
```

If you don't have write privileges on `~/ .bashrc`, you must always start the database with the `VERTICA_ADMINTOOLS_PASSTHROUGH` environment variable.

1. Always run the following command to start the database:

```
$ export VERTICA_ADMINTOOLS_PASSTHROUGH=MALLOC_ARENA_MAX=4 admintools -t start_db -d database_name
```

2. Verify on each node that MALLOC_ARENA_MAX is set for the process:

```
$ xargs --null --max-args=1 echo < /proc/$(pgrep vertica$)/environ | grep ARENA  
MALLOC_ARENA_MAX=4
```

Tuple Mover Operations

The Tuple Mover manages WOS and ROS data storage. It performs two operations:

- [Moveout](#) moves data from WOS to ROS. During moveout operations, the Tuple Mover also enforces storage policies for the storage location.
- [Mergeout](#) combines small ROS containers into larger ones and purges deleted data.

The Tuple Mover automatically performs these tasks in the background, at intervals that are set by its [configuration parameters](#).

Each of these operations occurs at different intervals across all nodes. The Tuple Mover runs independently on each node, ensuring that storage is managed appropriately even in the event of data skew.

Tuple Mover operations are automatic and transparent, and typically require no intervention. However, Vertica provides various ways to control Tuple Mover behavior. For details, see [Managing the Tuple Mover](#).

Mergeout

Mergeout is the Tuple Mover process that consolidates ROS containers and purges deleted records. Over time, the number of ROS containers increases enough to affect performance. It is then necessary to merge some of the ROS containers to avoid performance degradation. At that point, the Tuple Mover performs an automatic mergeout, combining two or more ROS containers into a single container. You can think of this process as a way of *defragmenting* the ROS.

Vertica keeps data from different partitions separate on disk. The Tuple Mover adheres to this separation policy when it consolidates ROS containers. Tuple Move does not merge ROS containers from different partitions. When a partition is first created, it typically has frequent data loads and requires regular activity from the Tuple Mover. As a partition ages, it commonly transitions to a mostly read-only workload and requires much less activity.

The Tuple Mover has two different policies for managing these different partition workloads:

- *Active partition* is the partition that was most recently created. The Tuple Mover uses a [STRATA](#) mergeout policy that keeps a collection of ROS container sizes to minimize the number of times any individual tuple is subjected to mergeout. The `ActivePartitionCount` parameter identifies how many partitions are being actively created.
- *Inactive partitions* are those that were not most recently created. The Tuple Mover consolidates the ROS containers to a minimal set while avoiding merging containers whose size exceeds `MaxMrgOutROSSizeMB`.

Partitions are not explicitly marked by the user as active or inactive; instead, the Tuple Mover uses the following algorithm to order the partitions from oldest to newest:

- If one partition was created before the other partition, it is older.
- If two partitions were created at the same time, but one partition was last updated earlier than the other partition, it is older.
- If two partitions were created and last updated at the same time, the partition with the smaller key is considered older.

If you perform a manual mergeout using the `DO_TM_TASK` function, *all* partitions are consolidated into the smallest possible number of containers, regardless of the value of the `ActivePartitionCount` parameter.

Mergeout Strata Algorithm

The mergeout operation uses a strata-based algorithm to verify that each tuple is subjected to a mergeout operation a small, constant number of times, despite the process used to load the data. The mergeout operation uses this algorithm to choose which ROS containers to merge for non-partitioned tables and for active partitions in partitioned tables.

Vertica builds strata for each active partition and for projections anchored to non-partitioned tables. The number of strata, the size of each stratum, and the maximum number of ROS containers in a stratum is computed based on disk size, memory, and the number of columns in a projection.

Merging small ROS containers before merging larger ones provides the maximum benefit during the mergeout process. The algorithm begins at stratum 0 and moves upward. It checks to see if the number of ROS containers in a stratum has reached a value equal to or greater than the maximum ROS containers allowed per stratum. The default value is 32. If the algorithm finds that a stratum is full, it marks the projections and the stratum as eligible for mergeout.

Mergeout of Deletion Markers

When you delete data from the database, Vertica does not remove it. Instead, it marks the data as deleted. Using many `DELETE` statements to mark a small number of rows relative to the size of a table can result in creating many small containers, the delete vectors, to hold data marked for deletion. Each delete vector container consumes resources, so a large number of such containers can impact performance, especially during recovery.

After the Tuple Mover performs a mergeout, it looks for deletion marker containers that hold few entries. If such containers exist, the Tuple Mover merges them together into a single, larger container. This process helps lower the overhead of tracking deleted data by freeing resources used by multiple, individual containers. The Tuple Mover does not purge or otherwise affect the deleted data, but consolidates delete vectors for greater efficiency.

Note: You can see the number and size of the containers holding the deletion marks by viewing the `V_MONITOR.DELETE_VECTORS` system table.

Automatic and Manual Mergeout

Vertica periodically checks ROS storage containers to determine whether delete vectors are eligible for purge, as follows:

1. Counts the number of 'aged-out' delete vectors in each container—that is, delete vectors that are as 'old' or older than the ancient history mark (AHM) epoch.
2. Calculates the percentage of aged-out delete vectors relative to the total number of records in the same ROS container.
3. If this percentage exceeds the threshold set by configuration parameter `PurgeMergeoutPercent` (by default, 20 percent), Vertica automatically performs a mergeout on the ROS container, and permanently removes all aged-out delete vectors from the ROS container.

You can also manually remove all aged-out delete vectors from ROS containers with two Vertica meta-functions:

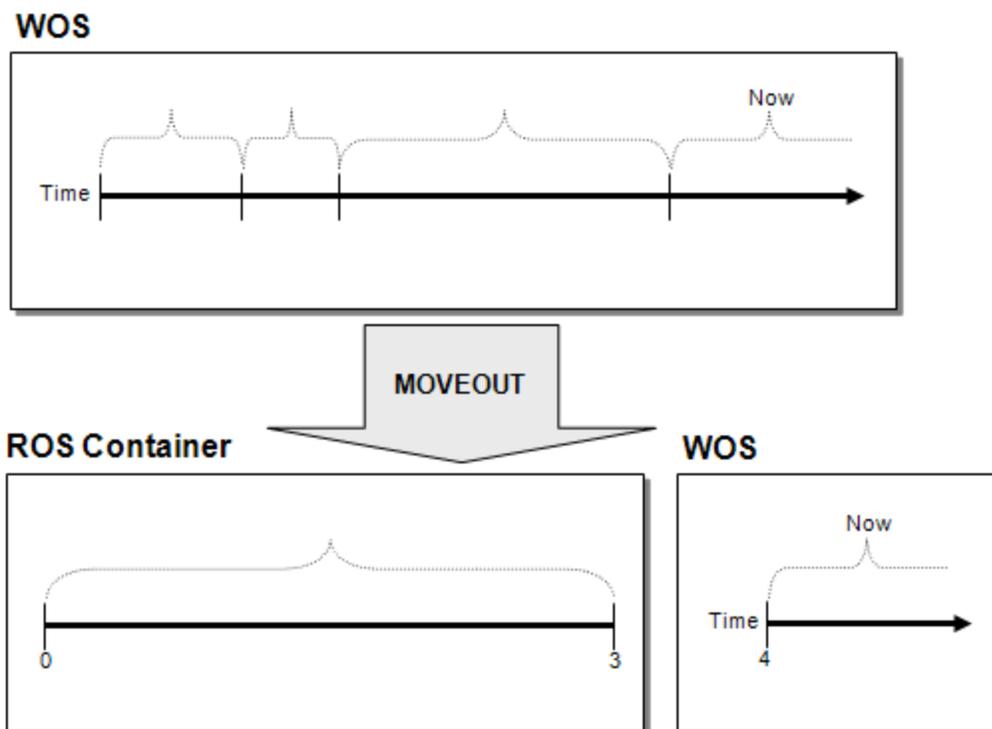
- `DO_TM_TASK('mergeout')`
- `PURGE`

Both functions remove all aged-out delete vectors from ROS containers, regardless of how many are in a given container.

Moveout

Moveout operations move data from memory (WOS) into a new ROS container. A moveout *flushes* all historical data from the WOS to the ROS.

The following illustration shows the effect of a projection moveout on a single node:



ROS Containers

A ROS (Read Optimized Store) container is a set of rows stored in a particular group of files. ROS containers are created by operations like Moveout or COPY DIRECT. You can query the `STORAGE_CONTAINERS` system table to see ROS containers. The ROS container layout can

differ across nodes due to data variance. Segmentation can deliver more rows to one node than another. Two data loads could fit in the WOS on one node and spill on another.

Managing the Tuple Mover

The Tuple Mover is preconfigured to handle typical workloads. However, some situations can require you to adjust Tuple Mover behavior. You can do so in various ways:

- [Configure resource pools](#)
- [Manage data loading](#)
- [Add threads](#)
- [Manage active data partitions](#)

Configuring Resource Pools

The Tuple Mover draws its resources from the TM resource pool. Adding more resources (RAM) to this pool, and changing its concurrency setting, can make the Tuple Mover more effective in dealing with high load rates.

The TM resource pool concurrency setting is determined by subtracting one from the value of `MAXCONCURRENCY`. This calculates the number of merges that can occur simultaneously through multiple threads. As a side effect of the concurrency setting, the Tuple Mover dedicates some threads to aggressively address small ROS containers, while other threads are reserved to work only on merges of ROS containers in the lower strata.

For the TM pool, `PLANNEDCONCURRENCY` must be proportional to the size of the RAM, the CPU, and the storage subsystem. Depending on the storage type, if you increase `PLANNEDCONCURRENCY` for the Tuple Mover threads, you might create a storage I/O bottleneck. Monitor the storage subsystem; if it becomes saturated with long I/O queues, more than two I/O queues, and long latency in read and write, adjust the `PLANNEDCONCURRENCY` parameter to keep the storage subsystem resources below saturation level. In addition, you might need to:

- Partition storage data files
- Adjust block-size optimization on storage subsystems such as RAID 5 or RAID 10
- Identify the optimal number of disks in the RAID array

The following statement illustrates how to increase the size of the TM resource pool and set the concurrency settings for the pool:

```
=> ALTER RESOURCE POOL tm MEMORYSIZE '4G' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

The WOSDATA resource pool settings also indirectly affect the Tuple Mover. In automatic mode, INSERT and COPY commands use the concurrency setting to determine whether data is small enough to store in WOS or if it should be written to ROS. Therefore, set this value to be the number of concurrent loads you expect to perform in your database. The WOSDATA resource pool also determines how much RAM the WOS can use.

```
=> ALTER RESOURCE POOL wosdata MAXMEMORYSIZE '4G' PLANNEDCONCURRENCY 3;
```

See [Managing Workloads](#) and [Resource Pool Architecture](#) in this guide and [ALTER RESOURCE POOL](#) and [Built-In Pools](#) in the SQL Reference Manual.

Managing Data Loads

By default, Vertica automatically decides whether the data should be placed in WOS or stored directly in ROS containers based on the amount of data processed by a COPY or INSERT command. Vertica stores large loads directly to disk and stores smaller loads in memory, which it later moves to disk.

For low-latency access to data, use small loads. The automatic Tuple Mover settings are the best option for handling such smaller loads. One exception is for single-node deployments, where a system failure would cause in-memory data to be lost. In this case, you might want to force all data loads to go directly to disk.

For high load rates, you might want the Tuple Mover to check for jobs more frequently by changing the `MergeOutInterval` and `MoveOutInterval` configuration parameters. Reduce the `MoveOutInterval` if you expect the peak load rate to fill the WOS quickly. Reduce `MergeOutInterval` if you anticipate performing many DIRECT loads or inserts.

In some cases, you might wish to control how Vertica loads data into ROS or WOS. Vertica provides various options for controlling how it loads data for specific tables and DML operations. This can also affect Tuple Mover behavior. For more information, see [Choosing a Load Method](#).

Adding Threads

If your database is receiving a large volume of data to load or if it is performing many DIRECT loads or inserts, consider allowing the Tuple Mover to perform more operations concurrently

by increasing the TM resource pool until it can keep up with anticipated peak loads. For example:

```
=> ALTER RESOURCE POOL TM MEMORYSIZE '4G' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

See [ALTER RESOURCE POOL](#) and [Built-In Pools](#) in the SQL Reference Manual.

Managing Active Data Partitions

By default, the Tuple Mover assumes that all loads and updates for partitioned tables are going to the same *active* partition. For example, if a table is partitioned by month, the Tuple Mover expects that after the start of a new month, no data is loaded into the partition for the prior month.

If loads and updates occur to more than one partition, set the `ActivePartitionCount` parameter to reflect the number of partitions that will be loading data. For example, if your database receives data for the current month as well as updates to the prior month, set `ActivePartitionCount` to 2. For tables partitioned by non-temporal attributes, set `ActivePartitionCount` to reflect the number of partitions that will be loaded simultaneously.

For more information, see [Managing Partitions](#).

See Also

- [Best Practices for Managing Workload Resources](#)

Managing Workloads

Vertica's resource management scheme allows diverse, concurrent workloads to run efficiently on the database. For basic operations, Vertica pre-configures the built-in [GENERAL pool](#) based on RAM and machine cores. You can customize the General pool to handle specific concurrency requirements.

You can also define new resource pools that you configure to limit memory usage, concurrency, and query priority. You can then optionally assign each database user to use a specific resource pool, which controls memory resources used by their requests.

User-defined pools are useful if you have competing resource requirements across different classes of workloads. Example scenarios include:

- A large batch job takes up all server resources, leaving small jobs that update a web page without enough resources. This can degrade user experience.

In this scenario, create a resource pool to handle web page requests and ensure users get resources they need. Another option is to create a limited resource pool for the batch job, so the job cannot use up all system resources.

- An application has lower priority than other applications and you want to limit the amount of memory and number of concurrent users for the low-priority application.

In this scenario, create a resource pool with an upper limit on the query's memory and associate the pool with users of the low-priority application.

You can also use resource pools to manage resources assigned to running queries. You can assign a run-time priority to a resource pool, as well as a threshold to assign different priorities to queries with different durations. See [Managing Resources At Query Run Time](#) for more information.

For detailed syntax of creating and managing resource pools see the following topics in the SQL Reference Manual:

Statements

- [ALTER RESOURCE POOL](#) alters a resource pool.
- [ALTER USER](#) associates a user with the RESOURCE POOL and MEMORYCAP parameters.
- [CREATE RESOURCE POOL](#) creates a resource pool.
- [CREATE USER](#) adds a name to the list of authorized database users and specifies that user's RESOURCE POOL and MEMORYCAP parameters.
- [DROP RESOURCE POOL](#) drops a user-created resource pool.
- [SET SESSION MEMORYCAP](#) sets the limit on amount of memory that any request issued by the session can consume.
- [SET SESSION RESOURCE POOL](#) associates a user session with specified resource pool.

System Tables

- [RESOURCE_ACQUISITIONS](#) provides details of resources (memory, open file handles, threads) acquired by each request for each resource pool in the system.
- [RESOURCE_POOL_DEFAULTS \(systab\)](#) lists default values for parameters in each internal and user-defined resource pool.
- [RESOURCE_POOL_STATUS](#) provides configuration settings of the various resource pools in the system, including internal pools.
- [RESOURCE_POOLS](#) displays information about the parameters the resource pool was configured with.
- [RESOURCE_QUEUES](#) provides information about requests pending for various resource pools.
- [RESOURCE_REJECTIONS](#) monitors requests for resources that are rejected by the Resource Manager.
- [RESOURCE_REJECTION_DETAILS](#) records an entry for each resource request that Vertica denies. This is useful for determining if there are resource space issues, as well as which users/pools encounter problems
- [SYSTEM_RESOURCE_USAGE](#) provides history about system resources, such as memory, CPU, network, disk, I/O.

Resource Manager

On a single-user environment, the system can devote all resources to a single query, getting the most efficient execution for that one query. More likely, your environment needs to run several queries at once, which can cause tension between providing each query the maximum amount of resources (fastest run time) and serving multiple queries simultaneously with a reasonable run time.

The Vertica Resource Manager (RM) lets you resolve this tension, while ensuring that every query is eventually serviced and that true system limits are respected at all times.

For example, when the system experiences resource pressure, the Resource Manager might queue queries until the resources become available or a timeout value is reached. In addition,

when you configure various RM settings, you can tune each query's target memory based on the expected number of concurrent queries running against the system.

This section discusses the detailed architecture and operation of the Resource Manager.

Resource Manager Impact on Query Execution

The Resource Manager (RM) impacts individual query execution in various ways. When a query is submitted to the database, the following series of events occur:

1. The query is parsed, optimized to determine an execution plan, and distributed to the participating nodes.
2. The Resource Manager is invoked on each node to estimate resources required to run the query and compare that with the resources currently in use. One of the following will occur:
 - If the memory required by the query alone would exceed the machine's physical memory, the query is rejected - it cannot possibly run. Outside of significantly under-provisioned nodes, this case is very unlikely.
 - If the resource requirements are not currently available, the query is queued. The query will remain on the queue until either sufficient resources are freed up and the query runs or the query times out and is rejected.
 - Otherwise the query is allowed to run.
3. The query starts running when all participating nodes allow it to run.

Note: Once the query is running, the Resource Manager further manages resource allocation using `RUNTIMEPRIORITY` and `RUNTIMEPRIORITYTHRESHOLD` parameters for the resource pool. See [Managing Resources At Query Run Time](#) for more information.

Apportioning resources for a specific query and the maximum number of queries allowed to run depends on the resource pool configuration. See [Resource Pool Architecture](#).

On each node, no resources are reserved or held while the query is in the queue. However, multi-node queries queued on some nodes will hold resources on the other nodes. Vertica makes every effort to avoid deadlocks in this situation.

Resource Pool Architecture

The Resource Manager handles resources as one or more resource pools, which are a pre-allocated subset of the system resources with an associated queue.

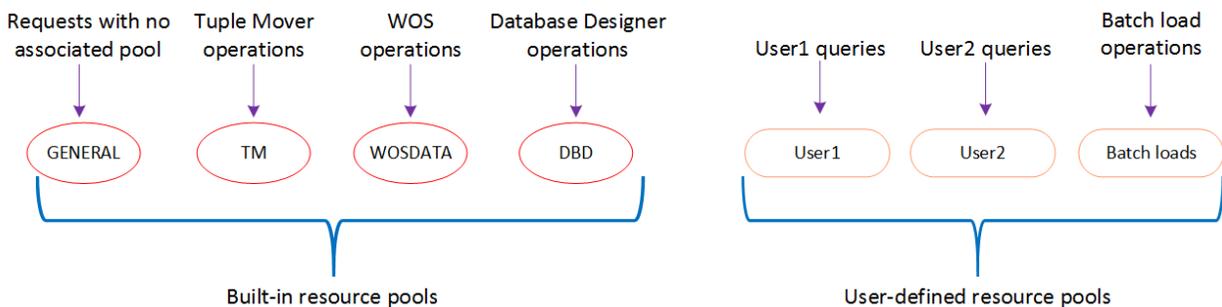
Vertica is preconfigured with a set of [Built-In Pools](#) that allocate resources to different request types, where the GENERAL pool allows for a certain concurrency level based on the RAM and cores in the machines.

Modifying and Creating Resource Pools

You can configure the built-in GENERAL pool based on actual concurrency and performance requirements, as described in [Built-In Pools](#). You can also create custom pools to handle various classes of workloads and optionally restrict user requests to your custom pools.

You create a pool using the [CREATE RESOURCE POOL](#) command.

You can create new resource pools and configure them for memory usage, concurrency, and queue priority. In addition, you can restrict a database user or user session to use a specific resource pool. Doing so allows you to control how memory, CPU, and other resources are allocated. If you have competing resource requirements across different classes of workloads, create user-defined pools. The following graphic illustrates what database operations are executed in which resource pool. Only four of the built-in pools are shown.



Defining Secondary Resource Pools

You can define secondary resource pools to which running queries can cascade if they exceed the initial pool's `RUNTIMECAP`.

Identifying a Secondary Pool

Defining secondary resource pools allows you to designate a place where queries that exceed the `RUNTIMECAP` of the pool on which they are running can execute. This way, if a query exceeds a pool's `RUNTIMECAP`, the query can cascade to a pool with a larger `RUNTIMECAP` instead of causing an error. When a query cascades to another pool, the original pool regains the memory used by that query.

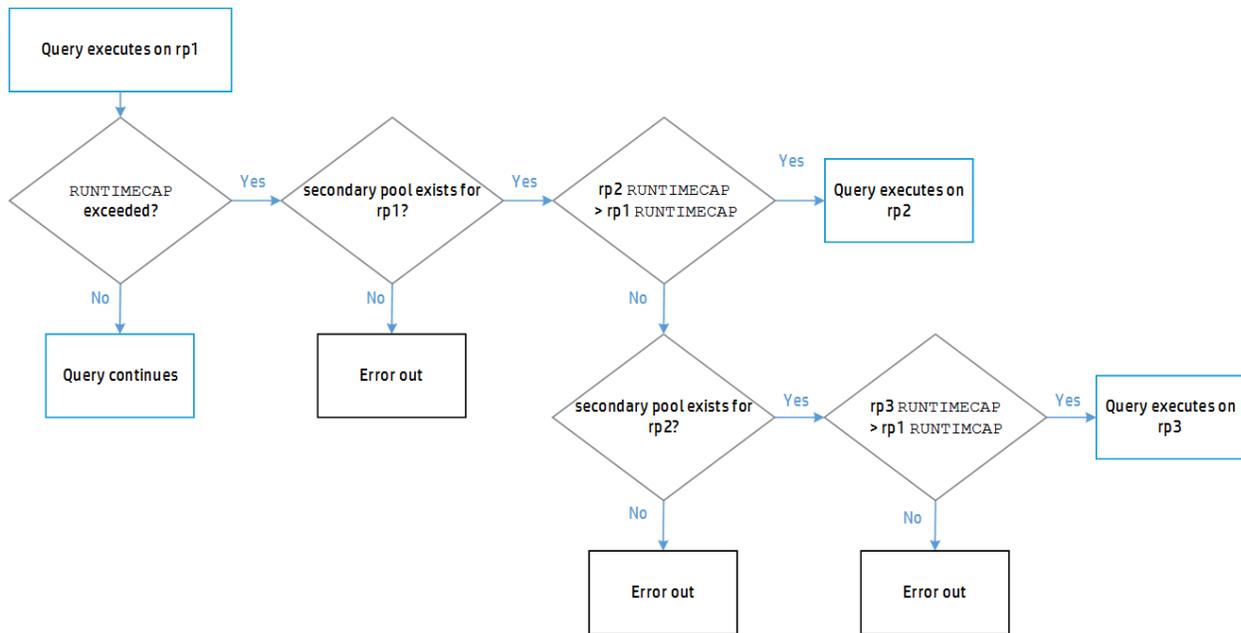
Because grant privileges are not considered on secondary pools, you can use this functionality to designate secondary resource pools for user queries without giving users explicit permission to run queries on that pool.

You can also use secondary pools as a place to store long-running queries for later. Using the `PRIORITY_HOLD` option, you can designate a secondary pool that re-queues the queries until `QUEUETIMEOUT` is reached or the pool's priority is changed to a non-hold value.

Query Cascade Path

Vertica routes queries to a secondary pool when the `RUNTIMECAP` on an initial pool is reached. Vertica then checks the secondary pool's `RUNTIMECAP` value. If the secondary pool's `RUNTIMECAP` is greater than the initial pool's value, the query executes on the secondary pool. If the secondary pool's `RUNTIMECAP` is less than or equal to the initial pool's value, Vertica retries the query on the next pool in the chain until it finds a pool on which the `RUNTIMECAP` is greater than the initial pool's value. If the secondary pool does not have sufficient resources available to execute the query at that time, `SELECT` queries may re-queue, re-plan, and abort on that pool. Other types of queries will fail due to insufficient resources. If no appropriate secondary pool exists for a query, the query will error out.

The following diagram demonstrates the path a query takes to execution.



Query Execution Time Allocation

After Vertica finds an appropriate pool on which to run the query, it continues to execute that query uninterrupted. The query now has the difference of the two pools' RUNTIMECAP limits in which to complete:

```
query execution time allocation = rp2 RUNTIMECAP - rp1 RUNTIMECAP
```

Using the CASCADE TO Parameter

As a superuser, you can identify the secondary pool by using the `CASCADE TO` parameter in the `CREATE RESOURCE POOL` or `ALTER RESOURCE POOL` statement. The secondary pool must already exist as a user-defined pool or the `GENERAL` pool. When using `CASCADE TO`, you cannot create a resource pool loop.

This example demonstrates a situation where the administrator wants `user1`'s queries to start on the `user_0` resource pool, but cascade to the `userOverflow` pool if the queries are too long.

```
=> CREATE RESOURCE POOL userOverflow RUNTIMECAP '5 minutes';
=> CREATE RESOURCE POOL user_0 RUNTIMECAP '1 minutes' CASCADE TO userOverflow;
=> CREATE USER "user1" RESOURCE POOL user_0;
```

In this scenario, `user1` cannot start his or her queries on the `userOverflow` resource pool, but because grant privileges are not considered for secondary pools, `user1`'s queries can cascade to the `userOverflow` pool if they exceed the `user_0` pool RUNTIMECAP. Using the secondary pool frees up space in the primary pool so short queries can run.

This example shows a situation where the administrator wants long-running queries to stay queued on a secondary pool.

```
=> CREATE RESOURCE POOL rp2 PRIORITY HOLD;  
=> CREATE RESOURCE POOL rp1 RUNTIMECAP '2 minutes' CASCADE TO rp2;  
=> SET SESSION RESOURCE_POOL = rp1;
```

In this scenario, queries that run on rp1 for more than 2 minutes will queue on rp2 until QUEUETIMEOUT is reached, at which point the queries will be rejected.

Dropping a Secondary Pool

If you try to drop a resource pool that is a secondary pool for another resource pool, Vertica returns an error. The error lists the resource pools that depend on the secondary pool you tried to drop. To drop a secondary resource pool, first set the `CASCADE TO` parameter to `DEFAULT` on the primary resource pool, and then drop the secondary pool.

For example, you can drop resource pool rp2, which is a secondary pool for rp1, as follows:

```
=> ALTER RESOURCE POOL rp1 CASCADE TO DEFAULT;  
=> DROP RESOURCE POOL rp2;
```

Parameter Considerations

The secondary resource pool's `CPUAFFINITYSET` and `CPUAFFINITYMODE` is applied to the query when it enters the pool.

The query adopts the secondary pool's `RUNTIMEPRIORITY` at different times, depending on the following circumstances:

- If the `RUNTIMEPRIORITYTHRESHOLD` timer was not started when the query was running in the primary pool, the query adopts the secondary resource pools' `RUNTIMEPRIORITY` when it cascades. This happens either when the `RUNTIMEPRIORITYTHRESHOLD` is not set for the primary pool or the `RUNTIMEPRIORITY` is set to `HIGH` for the primary pool.
- If the `RUNTIMEPRIORITYTHRESHOLD` was reached in the primary pool, the query adopts the secondary resource pools' `RUNTIMEPRIORITY` when it cascades.
- If the `RUNTIMEPRIORITYTHRESHOLD` was not reached in the primary pool and the secondary pool has no threshold, the query adopts the new pool's `RUNTIMEPRIORITY` when it cascades.
- If the `RUNTIMEPRIORITYTHRESHOLD` was not reached in the primary pool and the secondary pool has a threshold set.

- If the primary pool's `RUNTIMEPRIORITYTHRESHOLD` is greater than or equal to the secondary pool's `RUNTIMEPRIORITYTHRESHOLD`, the query adopts the secondary pool's `RUNTIMEPRIORITY` after the query reaches the `RUNTIMEPRIORITYTHRESHOLD` of the primary pool.

For example:

```
RUNTIMECAP of primary pool = 5 sec  
RUNTIMEPRIORITYTHRESHOLD of primary pool = 8 sec  
RUNTIMTPRIORITYTHRESHOLD of secondary pool = 7 sec
```

In this case, the query runs for 5 seconds on the primary pool and then cascades to the secondary pool. After another 3 seconds, 8 seconds total, the query adopts the `RUNTIMEPRIORITY` of the secondary pool.

- If the primary pool's `RUNTIMEPRIORITYTHRESHOLD` is less than the secondary pool's `RUNTIMEPRIORITYTHRESHOLD`, the query adopts the secondary pool's `RUNTIMEPRIORITY` after the query reaches the `RUNTIMEPRIORITYTHRESHOLD` of the secondary pool.

For example,

```
RUNTIMECAP of primary pool = 5 sec  
RUNTIMEPRIORITYTHRESHOLD of primary pool = 8 sec  
RUNTIMTPRIORITYTHRESHOLD of secondary pool = 12 sec
```

In this case, the query runs for 5 seconds on the primary pool and then cascades to the secondary pool. After another 7 seconds, 12 seconds total, the query adopts the `RUNTIMEPRIORITY` of the secondary pool.

Using Queries to Monitor Resource Pool Size and Usage

You can use the [Linux top command](#) to determine the overall CPU usage and I/O wait time across the system. However, because of file system caching, the resident memory size indicated by `top` is not a good indicator of actual memory use or available reserve.

Instead, Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests, and the state of the queues.

The [RESOURCE_POOLS](#) table lets you view various resource pools defined in the system (both internal and user-defined), and the [RESOURCE_POOL_STATUS](#) table lets you view the current state of the resource pools.

Examples

This example shows how to find the various resource pools defined in the system.

```
=> SELECT name, memorysize, maxmemorysize FROM V_CATALOG.RESOURCE_POOLS;
```

name	memorysize	maxmemorysize
general		Special: 95%
sysquery	64M	
sysdata	100M	10%
wosdata	0%	25%
tm	200M	
refresh	0%	
recovery	0%	
dbd	0%	
jvm	0%	10%

(9 rows)

Viewing Only User-Defined Resource Pools

To see only the user-defined resource pools, you can limit your query to return records where `IS_INTERNAL` is false.

Note: The user-defined pools shown in these examples also appear in subsequent sections related to Workload Management.

This example shows how to find information on user-defined resource pools:

```
=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency  
FROM V_CATALOG.RESOURCE_POOLS where is_internal = 'f';
```

name	memorysize	maxmemorysize	priority	maxconcurrency
load_pool	0%		10	
ceo_pool	250M		10	
ad hoc_pool	200M	200M	0	
billing_pool	0%		0	3
web_pool	25M		10	5
batch_pool	150M	150M	0	10
dept1_pool	0%		5	
dept2_pool	0%		8	

(8 rows)

Viewing the Status of All Resource Pools

The following example shows how to access the `V_MONITOR.RESOURCE_POOL_STATUS` table to return the current state of all resource pools on node0001:

```
=>\x
Expanded display is on
=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb,
       general_memory_borrowed_kb, running_query_count
       FROM V_MONITOR.RESOURCE_POOL_STATUS where node_name ilike '%node0001';

-[ RECORD 1 ]-----+-----
pool_name          | general
memory_size_kb     | 2983177
memory_size_actual_kb | 2983177
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 2 ]-----+-----
pool_name          | sysquery
memory_size_kb     | 65536
memory_size_actual_kb | 65536
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 3 ]-----+-----
pool_name          | sysdata
memory_size_kb     | 102400
memory_size_actual_kb | 102400
memory_inuse_kb    | 4096
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 4 ]-----+-----
pool_name          | wosdata
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 5 ]-----+-----
pool_name          | tm
memory_size_kb     | 204800
memory_size_actual_kb | 204800
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 6 ]-----+-----
pool_name          | refresh
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 7 ]-----+-----
pool_name          | recovery
memory_size_kb     | 0
```

```

memory_size_actual_kb      | 0
memory_inuse_kb            | 0
general_memory_borrowed_kb | 0
running_query_count        | 0
-[ RECORD 8 ]-----+-----
pool_name                   | dbd
memory_size_kb              | 0
memory_size_actual_kb      | 0
memory_inuse_kb            | 0
general_memory_borrowed_kb | 0
running_query_count        | 0
-[ RECORD 9 ]-----+-----
pool_name                   | jvm
memory_size_kb              | 0
memory_size_actual_kb      | 0
memory_inuse_kb            | 0
general_memory_borrowed_kb | 0
running_query_count        | 0

```

Viewing Query Resource Acquisitions

The following example displays all resources granted to the queries that are currently running. The information shown is stored in the `V_MONITOR.RESOURCE_ACQUISITIONS` table. You can see that the query execution used 708504 KB of memory from the GENERAL pool.

```

=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
       queue_entry_timestamp, acquisition_timestamp
       FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

-[ RECORD 1 ]-----+-----
pool_name           | sysquery
thread_count        | 4
open_file_handle_count | 0
memory_inuse_kb     | 4103
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05
acquisition_timestamp | 2013-12-05 07:07:08.815367-05
-[ RECORD 2 ]-----+-----
...
-[ RECORD 8 ]-----+-----
pool_name           | general
thread_count        | 12
open_file_handle_count | 18
memory_inuse_kb     | 708504
queue_entry_timestamp | 2013-12-04 12:55:38.566614-05
acquisition_timestamp | 2013-12-04 12:55:38.566623-05
-[ RECORD 9 ]-----+-----
...

```

You can determine how long a query waits in the queue before it can run. To do so, you obtain the difference between the `acquisition_timestamp` and the `queue_entry_timestamp` using a query as this example shows:

```
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,  
       (acquisition_timestamp-queue_entry_timestamp) AS 'queue wait'  
FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
```

```
-[ RECORD 1 ]-----+-----  
pool_name      | sysquery  
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05  
acquisition_timestamp | 2013-12-05 07:07:08.815367-05  
queue wait     | 00:00:00.000005  
-[ RECORD 2 ]-----+-----  
pool_name      | sysquery  
queue_entry_timestamp | 2013-12-05 07:07:14.714412-05  
acquisition_timestamp | 2013-12-05 07:07:14.714417-05  
queue wait     | 00:00:00.000005  
-[ RECORD 3 ]-----+-----  
pool_name      | sysquery  
queue_entry_timestamp | 2013-12-05 07:09:57.238521-05  
acquisition_timestamp | 2013-12-05 07:09:57.281708-05  
queue wait     | 00:00:00.043187  
-[ RECORD 4 ]-----+-----  
...
```

See Also

- See the [SQL Reference Manual](#) for detailed descriptions of the monitoring tables.
- See [Monitoring Resource Pools](#) for descriptions of other ways to monitor resource usage.

User Profiles

User profiles are attributes associated with a user that control that user's access to several system resources. These resources include:

- Resource pool to which a user is assigned (RESOURCE POOL)
- Maximum amount of memory a user's session can use (MEMORYCAP)
- Maximum amount of temporary file storage a user's session can use (TEMPSPACECAP)
- Maximum amount of time a user's query can run (RUNTIMECAP)

You can set these attributes with the [CREATE USER](#) statement and modify the attributes later with [ALTER USER](#).

Two strategies limit a user's access to resources: Setting attributes on the user directly to control resource use, or assigning the user to a resource pool. The first method lets you fine

tune individual users, while the second makes it easier to group many users together and set their collective resource usage.

The following examples illustrate how to set a user's resource pool attributes. For additional examples, see the scenarios described in [Using User-Defined Pools and User-Profiles for Workload Management](#).

Example

Set the user's RESOURCE POOL attribute to assign the user to a resource pool. To create a user named `user1` who has access to the resource pool `my_pool`, use the command:

```
=> CREATE USER user1 RESOURCE POOL my_pool;
```

To limit the amount of memory for a user without designating a pool, set the user's MEMORYCAP to either a particular unit or a percentage of the total memory available. For example, to create a user named `user2` whose sessions are limited to using 200 MBs memory each, use the command:

```
=> CREATE USER user2 MEMORYCAP '200M';
```

To limit the time a user's queries are allowed to run, set the RUNTIMECAP attribute. To prevent queries for `user2` from running more than five minutes, use this command:

```
=> ALTER USER user2 RUNTIMECAP '5 minutes';
```

To limit the amount of temporary disk space that the user's sessions can use, set the TEMPSPACECAP to either a particular size or a percentage of temporary disk space available. For example, the next statement creates `user3`, and limits her to using 1 GB of temporary space:

```
=> CREATE USER user3 TEMPSPACECAP '1G';
```

You can combine different attributes into a single command. For example, to limit the MEMORYCAP and RUNTIMECAP for `user3`, include both attributes in an [ALTER USER](#) statement:

```
=> ALTER USER user3 MEMORYCAP '750M' RUNTIMECAP '10 minutes';
ALTER USER
=> \x
Expanded display is on.
=> SELECT * FROM USERS;
-[ RECORD 1 ]-----+-----
user_id      | 45035996273704962
user_name    | release
is_super_user | t
resource_pool | general
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
```

```
run_time_cap      | unlimited
-[ RECORD 2 ]-----+-----
user_id           | 45035996273964824
user_name         | user1
is_super_user     | f
resource_pool     | my_pool
memory_cap_kb     | unlimited
temp_space_cap_kb | unlimited
run_time_cap      | unlimited
-[ RECORD 3 ]-----+-----
user_id           | 45035996273964832
user_name         | user2
is_super_user     | f
resource_pool     | general
memory_cap_kb     | 204800
temp_space_cap_kb | unlimited
run_time_cap      | 00:05
-[ RECORD 4 ]-----+-----
user_id           | 45035996273970230
user_name         | user3
is_super_user     | f
resource_pool     | general
memory_cap_kb     | 768000
temp_space_cap_kb | 1048576
run_time_cap      | 00:10
```

See Also

- [ALTER USER](#)
- [CREATE USER](#)

Target Memory Determination for Queries in Concurrent Environments

The resource pool parameters `MEMORYSIZE`, `MAXMEMORYSIZE`, and `PLANNEDCONCURRENCY` allow you to tune the target memory allocated to queries.

Note: For details about these parameters, see [CREATE RESOURCE POOL](#) in the SQL Reference Manual.

What is the Query Budget?

Each resource pool has a *query budget*, which is the target memory for queries executed on the associated pool. Vertica stores this value in the `query_budget_kb` column in the `V_MONITOR.RESOURCE_POOL_STATUS` system table.

Computing the Query Budget

The formula for computing a pool's query budget is as follows:

GENERAL pool:

- Query budget = Queuing threshold of the GENERAL pool / PLANNEDCONCURRENCY

If MEMORYSIZE is set to 0 and MAXMEMORYSIZE is NOT set:

- Query budget = Queuing threshold of the GENERAL pool / PLANNEDCONCURRENCY

If MEMORYSIZE is set to 0 and MAXMEMORYSIZE is set to a value other than the default:

- Query budget = Queuing threshold of the pool / PLANNEDCONCURRENCY

If MEMORYSIZE is set to a value other than the default:

- Query budget = MEMORYSIZE / PLANNEDCONCURRENCY of the pool

The consequence of this is by carefully tuning the MEMORYSIZE and PLANNEDCONCURRENCY parameters, it is possible to restrict the amount of memory used by a query to a desired size.

For a detailed example of query budget calculations, see [Do You Need to Put Your Query on a Budget?](#) in the [Vertica User Community](#).

Note: Vertica calculates the queuing threshold for a pool is 95% of a pool's MAXMEMORYSIZE.

Tuning the Query Budget

Normally, query budgets do not require any specific tuning. However, if you reduce the MAXMEMORYSIZE because you need memory for other purposes, be aware that you are also reducing the query budget. Reducing the query budget negatively impacts the query performance, particularly if the queries are complex.

To maintain the original query budget for the resource pool, if you reduce MAXMEMORYSIZE, be sure to also reduce the value of PLANNEDCONCURRENCY.

Parameter Values

The [RESOURCE_POOL_STATUS](#) system table contains the values for parameters for all Vertica resource pools.

See Also

- [User Profiles](#)

Managing Resources At Query Run Time

The Resource Manager estimates the resources required for queries to run, and then determines when to run queries and when to queue them.

The Resource Manager also lets you manage resources that are assigned to queries that are already running using either of these methods:

- [Setting Runtime Priority for the Resource Pool](#)--Use resource pool parameters to set the run time priority for queries running within the resource pool.
- [Changing Runtime Priority of a Running Query](#)--Manually change the run time priority of a running query.

Setting Runtime Priority for the Resource Pool

For each resource pool, you can manage resources that are assigned to queries that are already running. You assign each resource pool a runtime priority of HIGH, MEDIUM, or LOW. These settings determine the amount of runtime resources (such as CPU and I/O bandwidth) assigned to queries in the resource pool when they run. Queries in a resource pool with a HIGH priority are assigned greater runtime resources than those in resource pools with MEDIUM or LOW runtime priorities.

Prioritizing Queries Within a Resource Pool

While runtime priority helps to manage resources for the resource pool, there may be instances where you want some flexibility within a resource pool. For instance, you may want to ensure that very short queries run at a high priority, while also ensuring that all other queries run at a medium or low priority.

The Resource Manager allows you this flexibility by letting you set a *runtime priority threshold* for the resource pool. With this threshold, you specify a time limit (in seconds) by which a query must finish before it is assigned the runtime priority of the resource pool. All queries begin running with a HIGH priority; once a query's duration exceeds the time limit specified in the runtime priority threshold, it is assigned the runtime priority of the resource pool.

Setting Runtime Priority and Runtime Priority Threshold

You specify runtime priority and runtime priority threshold by setting two resource pool parameters with [CREATE RESOURCE POOL](#) or [ALTER RESOURCE POOL](#):

- `RUNTIMEPRIORITY`
- `RUNTIMEPRIORITYTHRESHOLD`

Changing Runtime Priority of a Running Query

[CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY](#) lets you to change a query's runtime priority. You can change the runtime priority of a query that is already executing.

This function takes two arguments:

- The query's transaction ID, obtained from the system table [SESSIONS](#)
- The desired priority, one of the following string values: HIGH, MEDIUM, or LOW

Restrictions

Superusers can change the runtime priority of any query to any priority level. The following restrictions apply to other users:

- They can only change the runtime priority of their own queries.
- They cannot raise the runtime priority of a query to a level higher than that of the resource pools.

Procedure

Changing a query's runtime priority is a two-step procedure:

1. Get the query's transaction ID by querying the system table [SESSIONS](#). For example, the following statement returns information about all running queries:

```
=> SELECT transaction_id, runtime_priority, transaction_description from SESSIONS;
```

2. Run [CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY](#), specifying the query's

transaction ID and desired runtime priority:

```
=> SELECT CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY(45035996273705748, 'low')
```

Manually Moving Queries to Different Resource Pools

If you are the database administrator, you can move queries to another resource pool mid-execution using the [MOVE_STATEMENT_TO_RESOURCE_POOL](#) meta-function.

You might want to use this feature if a single query is using a large amount of resources, preventing smaller queries from executing.

What Happens When a Query Moves to a Different Resource Pool

When a query is moved from one resource pool to another, it continues executing, provided the target pool has enough resources to accommodate the incoming query. If sufficient resources cannot be assigned in the target pool on at least one node, Vertica cancels the query and attempts to re-plan the query. If Vertica cannot re-plan the query, the query is canceled indefinitely.

When you successfully move a query to a target resource pool, its resources will be accounted for by the target pool and released on the first pool.

If you move a query to a resource pool with `PRIORITY HOLD`, Vertica cancels the query and queues it on the target pool. This cancellation remains in effect until you change the `PRIORITY` or move the query to another pool without `PRIORITY HOLD`. You can use this option if you want to store long-running queries for later use.

You can view the [RESOURCE_ACQUISITIONS](#) or [RESOURCE_POOL_STATUS](#) system tables to determine if the target pool can accommodate the query you want to move. Be aware that the system tables may change between the time you query the tables and the time you invoke the `MOVE_STATEMENT_TO_RESOURCE_POOL` meta-function.

When a query successfully moves from one resource pool to another mid-execution, it executes until the greater of the existing and new `RUNTIMECAP` is reached. For example, if the `RUNTIMECAP` on the initial pool is greater than that on the target pool, the query can execute until the initial `RUNTIMECAP` is reached.

When a query successfully moves from one resource pool to another mid-execution the CPU affinity will change.

Using the MOVE_STATEMENT_TO_RESOURCE_POOL Function

To manually move a query from its current resource pool to another resource pool, use the `MOVE_STATEMENT_TO_RESOURCE_POOL` meta-function. Provide the session id, transaction id, statement id, and target resource pool name, as shown:

```
=> SELECT MOVE_STATEMENT_TO_RESOURCE_POOL ('v_vmart_node0001.example.-31427:0x82fbm',  
45035996273711993, 1, 'my_target_pool');
```

See Also:

- [Defining Secondary Resource Pools](#)
- [MOVE_STATEMENT_TO_RESOURCE_POOL](#)
- [RESOURCE_POOL_MOVE](#)

Restoring Resource Manager Defaults

The system table `V_CATALOG.RESOURCE_POOL_DEFAULTS` stores default values for all parameters for all built-in and user-defined resource pools.

If you have changed the value of any parameter in any of your resource pools and want to restore it to its default, you can simply alter the table and set the parameter to `DEFAULT`. For example, the following statement sets the `RUNTIMEPRIORITY` for the resource pool `sysquery` back to its default value:

```
VMart=> ALTER RESOURCE POOL sysquery RUNTIMEPRIORITY DEFAULT;
```

See Also

- [RESOURCE_POOL_DEFAULTS](#)

Best Practices for Managing Workload Resources

This section provides general guidelines and best practices on how to set up and tune resource pools for various common scenarios.

Note: The exact settings for resource pool parameters are heavily dependent on your query mix, data size, hardware configuration, and concurrency requirements. VERTICA recommends performing your own experiments to determine the optimal configuration for your system.

Basic Principles for Scalability and Concurrency Tuning

An Vertica database runs on a cluster of commodity hardware. All loads and queries running against the database take up system resources, such as CPU, memory, disk I/O bandwidth, file handles, and so forth. The performance (run time) of a given query depends on how much resource it has been allocated.

When running more than one query concurrently on the system, both queries are sharing the resources; therefore, each query could take longer to run than if it was running by itself. In an efficient and scalable system, if a query takes up all the resources on the machine and runs in X time, then running two such queries would double the run time of each query to $2X$. If the query runs in $> 2X$, the system is not linearly scalable, and if the query runs in $< 2X$ then the single query was wasteful in its use of resources. Note that the above is true as long as the query obtains the minimum resources necessary for it to run and is limited by CPU cycles. Instead, if the system becomes bottlenecked so the query does not get enough of a particular resource to run, then the system has reached a limit. In order to increase concurrency in such cases, the system must be expanded by adding more of that resource.

In practice, Vertica should achieve near linear scalability in run times, with increasing concurrency, until a system resource limit is reached. When adequate concurrency is reached without hitting bottlenecks, then the system can be considered as ideally sized for the workload.

Note: Typically Vertica queries on segmented tables run on multiple (likely all) nodes of the cluster. Adding more nodes generally improves the run time of the query almost linearly.

Setting a Runtime Limit for Queries

You can set a limit for the amount of time a query is allowed to run. You can set this limit at three levels, listed in descending order of precedence:

1. The resource pool to which the user is assigned.
2. User profile with RUNTIMECAP configured by [CREATE USER/ALTER USER](#)
3. Session queries, set by [SET SESSION RUNTIMECAP](#)

In all cases, you set the runtime limit with an [interval](#) value that does not exceed one year. When you set runtime limit at multiple levels, Vertica always uses the shortest value. If a runtime limit is set for a non-superuser, that user cannot set any session to a longer runtime limit. Superusers can set the runtime limit for other users and for their own sessions, to any value up to one year, inclusive.

Example

user1 is assigned to the ad_hoc_queries resource pool:

```
=> CREATE USER user1 RESOURCE POOL ad_hoc_queries;
```

RUNTIMECAP for user1 is set to 1 hour:

```
=> ALTER USER user1 RUNTIMECAP '60 minutes';
```

RUNTIMECAP for the ad_hoc_queries resource pool is set to 30 minutes:

```
=> ALTER RESOURCE POOL ad_hoc_queries RUNTIMECAP '30 minutes';
```

In this example, Vertica terminates user1's queries if they exceed 30 minutes. Although the user1's runtime limit is set to one hour, the pool on which the query runs, which has a 30-minute runtime limit, has precedence.

Note: If a secondary pool for the ad_hoc_queries pool is specified using the CASCADE TO function, the query executes on that pool when the RUNTIMECAP on the ad_hoc_queries pool is surpassed.

See Also

- [RESOURCE_POOLS](#)
- [Defining Secondary Resource Pools](#)

Handling Session Socket Blocking

A session socket can be blocked while awaiting client input or output for a given query. Session sockets are typically blocked for numerous reasons—for example, when the Vertica execution engine transmits data to the client, or a [COPY LOCAL](#) operation awaits load data from the client.

In rare cases, a session socket can remain indefinitely blocked. For example, a query times out on the client, which tries to forcibly cancel the query, or relies on the session [RUNTIMECAP setting](#) to terminate it. In either case, if the query ends while awaiting messages or data, the socket can remain blocked and the session hang until it is forcibly closed.

Configuring a Grace Period

You can configure the system with a grace period, during which a lagging client or server can catch up and deliver a pending response. If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated. If no grace period is set, the query can maintain its block on the socket indefinitely.

You should set the session grace period high enough to cover an acceptable range of latency and avoid closing sessions prematurely—for example, normal client-side delays in responding to the server. Very large load operations might require you to adjust the session grace period as needed.

You can set the grace period at four levels, listed in descending order of precedence:

1. Session (highest)
2. User
3. Node
4. Database

Setting Grace Periods for the Database and Nodes

At the database and node levels, you set the grace period to any [interval](#) up to 20 days, through configuration parameter `BlockedSocketGracePeriod`:

- `ALTER DATABASE db-name SET BlockedSocketGracePeriod = 'interval';`
- `ALTER NODE node-name SET BlockedSocketGracePeriod = 'interval';`

By default, the grace period for both levels is set to an empty string, which allows unlimited blocking.

Setting Grace Periods for Users and Sessions

You can set the grace period for individual users and for a given session, as follows:

- `{ CREATE | ALTER USER } user-name GRACEPERIOD { 'interval' | NONE };`
- `SET SESSION GRACEPERIOD { 'interval' | = DEFAULT | NONE };`

A user can set a session to any interval equal to or less than the grace period set for that user. Superusers can set the grace period for other users, and for their own sessions, to any value up to 20 days, inclusive.

Examples

Superuser `dbadmin` sets the database grace period to 6 hours. This limit only applies to non-superusers. `dbadmin` can set the session grace period for herself to any value up to 20 days—in this case, 10 hours:

```
=> ALTER DATABASE VMart SET BlockedSocketGracePeriod = '6 hours';
ALTER DATABASE
=> SHOW CURRENT BlockedSocketGracePeriod;
  level |          name          | setting
-----+-----+-----
DATABASE | BlockedSocketGracePeriod | 6 hours
(1 row)

=> SET SESSION GRACEPERIOD '10 hours';
SET
=> SHOW GRACEPERIOD;
  name      | setting
-----+-----
graceperiod | 10:00
(1 row)
```

dbadmin creates user user777 created with no grace period setting. Thus, the effective grace period for user777 is derived from the database setting of BlockedSocketGracePeriod, which is 6 hours. Any attempt by user777 to set the session grace period to a value greater than 6 hours returns with an error:

```
=> CREATE USER user777;
=> \c - user777
You are now connected as user "user777".
=> SHOW GRACEPERIOD;
   name      | setting
-----+-----
 graceperiod | 06:00
(1 row)

=> SET SESSION GRACEPERIOD '7 hours';
ERROR 8175: The new period 07:00 would exceed the database limit of 06:00
```

dbadmin sets a grace period of 5 minutes for user777. Now, user777 can set the session grace period to any value equal to or less than the user-level setting:

```
=> \c
You are now connected as user "dbadmin".
=> ALTER USER user777 GRACEPERIOD '5 minutes';
ALTER USER
=> \c - user777
You are now connected as user "user777".
=> SET SESSION GRACEPERIOD '6 minutes';
ERROR 8175: The new period 00:06 would exceed the user limit of 00:05
=> SET SESSION GRACEPERIOD '4 minutes';
SET
```

Using User-Defined Pools and User-Profiles for Workload Management

The scenarios in this section describe common workload-management issues, and provide solutions with examples.

Periodic Batch Loads

Scenario

You do batch loads every night, or occasionally (infrequently) during the day. When loads are running, it is acceptable to reduce resource usage by queries, but at all other times you want all resources to be available to queries.

Solution

Create a separate resource pool for loads with a higher priority than the preconfigured setting on the build-in GENERAL pool.

In this scenario, nightly loads get preference when borrowing memory from the GENERAL pool. When loads are not running, all memory is automatically available for queries.

Note: If you are using the WOS, tune the PLANNEDCONCURRENCY parameter of the WOSDATA pool to the number of concurrent loads. This ensures that AUTO spill to ROS is configured in an optimal fashion.

Example

Create a resource pool with the `PRIORITY` of the pool set higher than the `GENERAL` pool.

For example, to create a pool designated for loads that has a higher priority than the `GENERAL` pool, set `load_pool` with a priority of 10:

```
=> CREATE RESOURCE POOL load_pool PRIORITY 10;
```

Edit the `WOSDATA` pool `PLANNEDCONCURRENCY`:

```
=> ALTER RESOURCE POOL WOSDATA PLANNEDCONCURRENCY 6;
```

Modify the user's resource pool:

```
=> ALTER USER load_user RESOURCE POOL load_pool;
```

CEO Query

Scenario

The CEO runs a report every Monday at 9AM, and you want to be sure that the report always runs.

Solution

To ensure that a certain query or class of queries always gets resources, you could create a dedicated pool for it as follows:

1. Using the [PROFILE](#) command, run the query that the CEO runs every week to determine how much memory should be allocated:

```
=> PROFILE SELECT DISTINCT s.product_key, p.product_description
-> FROM store.store_sales_fact s, public.product_dimension p
-> WHERE s.product_key = p.product_key AND s.product_version = p.product_version
-> AND s.store_key IN (
->   SELECT store_key FROM store.store_dimension
->   WHERE store_state = 'MA')
-> ORDER BY s.product_key;
```

2. At the end of the query, the system returns a notice with resource usage:

```
NOTICE: Statement is being profiled.HINT: select * from v_monitor.execution_engine_profiles
where
transaction_id=45035996273751349 and statement_id=6;
NOTICE: Initiator memory estimate for query: [on pool general: 1723648 KB,
minimum: 355920 KB]
```

3. Create a resource pool with MEMORYSIZE reported by the above hint to ensure that the CEO query has at least this memory reserved for it:

```
=> CREATE RESOURCE POOL ceo_pool MEMORYSIZE '1800M' PRIORITY 10;
CREATE RESOURCE POOL
=> \x
Expanded display is on.
=> SELECT * FROM resource_pools WHERE name = 'ceo_pool';
-[ RECORD 1 ]-----+-----
name                | ceo_pool
is_internal          | f
memorysize           | 1800M
maxmemorysize        |
priority             | 10
queuetimeout         | 300
plannedconcurrency   | 4
maxconcurrency       |
singleinitiator      | f
```

4. Assuming the CEO report user already exists, associate this user with the above resource pool using ALTER USER statement.

```
=> ALTER USER ceo_user RESOURCE POOL ceo_pool;
```

5. Issue the following command to confirm that the `ceo_user` is associated with the `ceo_pool`:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';  
-[ RECORD 1 ]-----  
user_id      | 45035996273713548  
user_name    | ceo_user  
is_super_user | f  
resource_pool | ceo_pool  
memory_cap_kb | unlimited
```

If the CEO query memory usage is too large, you can ask the Resource Manager to reduce it to fit within a certain budget. See [Target Memory Determination for Queries in Concurrent Environments](#).

Preventing Runaway Queries

Scenario

Joe, a business analyst often runs big reports in the middle of the day that take up the whole machine's resources. You want to prevent Joe from using more than 100MB of memory, and you want to also limit Joe's queries to run for less than 2 hours.

Solution

[User Profiles](#) provides a solution to this scenario. To restrict the amount of memory Joe can use at one time, set a MEMORYCAP for Joe to 100MB using the [ALTER USER](#) command. To limit the amount of time that Joe's query can run, set a RUNTIMECAP to 2 hours using the same command. If any query run by Joe takes up more than its cap, Vertica rejects the query.

If you have a whole class of users whose queries you need to limit, you can also create a resource pool for them and set RUNTIMECAP for the resource pool. When you move these users to the resource pool, Vertica limits all queries for these users to the RUNTIMECAP you specified for the resource pool.

Example

```
=> ALTER USER analyst_user MEMORYCAP '100M' RUNTIMECAP '2 hours';
```

If Joe attempts to run a query that exceeds 100MB, the system returns an error that the request exceeds the memory session limit, such as the following example:

```
\i vmart_query_04.sqlvsq1:vmart_query_04.sql:12: ERROR: Insufficient resources to initiate plan  
on pool general [Request exceeds memory session limit: 137669KB > 102400KB]
```

Only the system database administrator (dbadmin) can increase only the MEMORYCAP setting. Users cannot increase their own MEMORYCAP settings and will see an error like the following if they attempt to edit their MEMORYCAP or RUNTIMECAP settings:

```
ALTER USER analyst_user MEMORYCAP '135M';  
ROLLBACK: permission denied
```

Restricting Resource Usage of Ad Hoc Query Application

Scenario

You recently made your data warehouse available to a large group of users who are not experienced SQL users. Some of the users run reports that operate on a large number of rows and overwhelm the system. You want to throttle usage of the system by these users.

Solution

The simplest solution is to create a standalone resource pool for the ad hoc applications so that the total MEMORYSIZE is fixed. Recall that in a standalone pool, MAXMEMORYSIZE is set equal to MEMORYSIZE so no memory can be borrowed from the GENERAL pool. Associate this user pool with database users from which the application uses to connect to the database. Also set RUNTIMECAP to limit the maximum duration of an ad hoc query.

Other solutions include limiting the memory usage of individual users such as in the [Preventing Runaway Queries](#).

Example

To create a standalone resource pool for the ad hoc users, set the MEMORYSIZE equal to the MAXMEMORYSIZE:

```
=> CREATE RESOURCE POOL adhoc_pool MEMORYSIZE '200M' MAXMEMORYSIZE '200M' PRIORITY 0 QUEUETIMEOUT
300 PLANNEDCONCURRENCY 4;
=> SELECT pool_name, memory_size_kb, queueing_threshold_kb
FROM V_MONITOR.RESOURCE_POOL_STATUS w
WHERE is_standalone = 'true' AND is_internal = 'false';
pool_name | memory_size_kb | queueing_threshold_kb
-----+-----+-----
adhoc_pool |          204800 |             153600
(1 row)
```

After the pool has been created, associate the ad hoc users with the adhoc_pool:

```
=> ALTER USER app1_user RESOURCE POOL adhoc_pool;
=> ALTER RESOURCE POOL adhoc_pool MEMORYSIZE '10M' MAXMEMORYSIZE '10M';
\i vmart_query_04.sql
vsq1:vmart_query_04.sql:12: ERROR: Insufficient resources to initiate plan
on pool adhoc_pool [Request Too Large:Memory(KB)
Exceeded: Requested = 84528, Free = 10240 (Limit = 10240, Used = 0)]
```

The query will not borrow memory from the GENERAL pool and gets rejected with a 'Request Too Large' message.

Setting a Hard Limit on Concurrency for an Application

Scenario

For billing purposes, analyst Jane would like to impose a hard limit on concurrency for this application. How can she achieve this?

Solution

The simplest solution is to create a separate resource pool for the users of that application and set its `MAXCONCURRENCY` to the desired concurrency level. Any queries beyond `MAXCONCURRENCY` are queued.

Tip: Micro Focus recommends leaving `PLANNEDCONCURRENCY` to the default level so the queries get their maximum amount of resources. The system as a whole thus runs with the highest efficiency.

Example

In this example, there are four billing users associated with the billing pool. The objective is to set a hard limit on the resource pool so a maximum of three concurrent queries can be executed at one time. All other queries will queue and complete as resources are freed.

```
=> CREATE RESOURCE POOL billing_pool MAXCONCURRENCY 3 QUEUETIMEOUT 2;
=> CREATE USER bill1_user RESOURCE POOL billing_pool;
=> CREATE USER bill2_user RESOURCE POOL billing_pool;
=> CREATE USER bill3_user RESOURCE POOL billing_pool;
=> CREATE USER bill4_user RESOURCE POOL billing_pool;
=> \x
Expanded display is on.

=> select maxconcurrency,queuetimeout from resource_pools where name = 'billing_pool';
maxconcurrency | queuetimeout
-----+-----
                3 |                2
(1 row)
> SELECT reason, resource_type, rejection_count FROM RESOURCE_REJECTIONS
WHERE pool_name = 'billing_pool' AND node_name ilike '%node0001';
reason | resource_type | rejection_count
-----+-----+-----
Timeout waiting for resource request | Queries | 16
(1 row)
```

If queries are running and do not complete in the allotted time (default timeout setting is 5 minutes), the next query requested gets an error similar to the following:

```
ERROR: Insufficient resources to initiate plan on pool billing_pool [Timeout waiting for resource
request: Request exceeds limits:
Queries Exceeded: Requested = 1, Free = 0 (Limit = 3, Used = 3)]
```

The table below shows that there are three active queries on the billing pool.

```
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb FROM RESOURCE_ACQUISITIONS
WHERE pool_name = 'billing_pool';
pool_name | thread_count | open_file_handle_count | memory_inuse_kb
-----+-----+-----+-----
billing_pool | 4 | 5 | 132870
billing_pool | 4 | 5 | 132870
billing_pool | 4 | 5 | 132870
(3 rows)
```

Handling Mixed Workloads: Batch versus Interactive

Scenario

You have a web application with an interactive portal. Sometimes when IT is running batch reports, the web page takes a long time to refresh and users complain, so you want to provide a better experience to your web site users.

Solution

The principles learned from the previous scenarios can be applied to solve this problem. The basic idea is to segregate the queries into two groups associated with different resource pools. The prerequisite is that there are two distinct database users issuing the different types of queries. If this is not the case, do consider this a best practice for application design.

Method 1

Create a dedicated pool for the web page refresh queries where you:

- Size the pool based on the average resource needs of the queries and expected number of concurrent queries issued from the portal.
- Associate this pool with the database user that runs the web site queries. See [CEO Query](#) for information about creating a dedicated pool.

This ensures that the web site queries always run and never queue behind the large batch jobs. Leave the batch jobs to run off the GENERAL pool.

For example, the following pool is based on the average resources needed for the queries running from the web and the expected number of concurrent queries. It also has a higher PRIORITY to the web queries over any running batch jobs and assumes the queries are being tuned to take 250M each:

```
=> CREATE RESOURCE POOL web_pool
    MEMORYSIZE '250M'
    MAXMEMORYSIZE NONE
    PRIORITY 10
    MAXCONCURRENCY 5
    PLANNEDCONCURRENCY 1;
```

Method 2

Create a standalone pool to limit the batch reports down to a fixed memory size so memory is always left available for other purposes. For details, see [Restricting Resource Usage of Ad Hoc Query Application](#).

For example:

```
=> CREATE RESOURCE POOL batch_pool
    MEMORYSIZE '4G'
    MAXMEMORYSIZE '4G'
    MAXCONCURRENCY 10;
```

The same principle can be applied if you have three or more distinct classes of workloads.

Setting Priorities on Queries Issued By Different Users

Scenario

You want user queries from one department to have a higher priority than queries from another department.

Solution

The solution is similar to the [mixed workload case](#). In this scenario, you do not limit resource usage; you set different priorities. To do so, create two different pools, each with MEMORYSIZE=0% and a different PRIORITY parameter. Both pools borrow from the GENERAL pool, however when competing for resources, the priority determine the order in which each pool's request is granted. For example:

```
=> CREATE RESOURCE POOL dept1_pool PRIORITY 5;  
=> CREATE RESOURCE POOL dept2_pool PRIORITY 8;
```

If you find this solution to be insufficient, or if one department's queries continuously starves another department's users, you can add a reservation for each pool by setting MEMORYSIZE so some memory is guaranteed to be available for each department.

For example, both resources use the GENERAL pool for memory, so you can allocate some memory to each resource pool by using ALTER RESOURCE POOL to change MEMORYSIZE for each pool:

```
=> ALTER RESOURCE POOL dept1_pool MEMORYSIZE '100M';  
=> ALTER RESOURCE POOL dept2_pool MEMORYSIZE '150M';
```

Continuous Load and Query

Scenario

You want your application to run continuous load streams, but many have up concurrent query streams. You want to ensure that performance is predictable.

Solution

The solution to this scenario depends on your query mix. In all cases, the following approach applies:

1. Determine the number of continuous load streams required. This may be related to the desired load rate if a single stream does not provide adequate throughput, or may be more directly related to the number of sources of data to load. Also determine if automatic storage is best, or if DIRECT is required. Create a dedicated resource pool for the loads, and associate it with the database user that will perform them. See [CREATE RESOURCE POOL](#) for details.

In general, concurrency settings for the load pool should be less than the number of cores per node. Unless the source processes are slow, it is more efficient to dedicate more memory per load, and have additional loads queue. Adjust the load pool's QUEUETIMEOUT setting if queuing is expected.

2. If using automatic targeting of [COPY](#) and [INSERT](#), set the PLANNEDCONCURRENCY parameter of the WOSDATA pool to the number of concurrent loads expected. Also, set MEMORYSIZE of the WOS to the expected size of the loaded data to ensure that small loads don't spill to ROS immediately. See [Built-In Pools](#) for details.
3. Run the load workload for a while and observe whether the load performance is as expected. If the Tuple Mover is not tuned adequately to cover the load behavior, see [Managing the Tuple Mover](#) in Administrator's Guide.
4. If there is more than one kind of query in the system—for example, some queries must be answered quickly for interactive users, while others are part of a batch reporting process—follow the guidelines in [Handling Mixed Workloads: Batch versus Interactive](#).
5. Let the queries run and observe performance. If some classes of queries do not perform as desired, then you might need to tune the GENERAL pool as outlined in [Restricting Resource Usage of Ad Hoc Query Application](#), or create more dedicated resource pools for those queries. See more information, see [CEO Query](#) and [Handling Mixed Workloads: Batch versus Interactive](#).

See the sections on [Managing Workloads](#) and [CREATE RESOURCE POOL](#) for information on obtaining predictable results in mixed workload environments.

Prioritizing Short Queries at Run Time

Scenario

You recently created a resource pool for users who are not experienced with SQL and who frequently run ad hoc reports. You have managed resource allocation by creating a standalone resource pool that will prevent these queries from borrowing resources from the GENERAL pool, but now you want to manage resources at run time and ensure that short queries always run with a high priority and are never queued as a result of limited run-time resources.

Solution

Set the `RUNTIMEPRIORITY` for the resource pool to `MEDIUM` or `LOW`. Set the `RUNTIMEPRIORITYTHRESHOLD` for the resource pool to the duration of queries you want to ensure always run at a high priority. For instance, if you set this value to 5, all queries that complete within 5 seconds will run at high priority. Any other query that exceeds 5 seconds will drop down to the `RUNTIMEPRIORITY` assigned to the resource pool (`MEDIUM` or `LOW`).

Example

To ensure that all queries with a duration of less than 5 seconds always run at a high priority, modify `adhoc_pool` as follows:

- Set the `RUNTIMEPRIORITY` to `MEDIUM`
- Set the `RUNTIMETHRESHOLD` to 5

```
=> ALTER RESOURCE POOL ad_hoc_pool RUNTIMEPRIORITY medium RUNTIMEPRIORITYTHRESHOLD 5;
```

Dropping the Runtime Priority of Long Queries

Scenario

You want most queries in a resource pool to run at a HIGH runtime priority; however, you'd like to be able to drop jobs longer than 1 hour to a lower priority.

Solution

Set the `RUNTIMEPRIORITY` for the resource pool to `LOW` and set the `RUNTIMEPRIORITYTHRESHOLD` to a number that cuts off only the longest jobs.

Example

To ensure that all queries with a duration of more than 3600 seconds (1 hour) are assigned a low runtime priority, modify the resource pool as follows:

- Set the `RUNTIMEPRIORITY` to `LOW`.
- Set the `RUNTIMETHRESHOLD` to `3600`

```
=> ALTER RESOURCE POOL ad_hoc_pool RUNTIMEPRIORITY low RUNTIMEPRIORITYTHRESHOLD 3600;
```

Tuning Built-In Pools

The scenarios in this section describe how to tune built-in pools.

- [Restricting Vertica to Take Only 60% of Memory](#)
- [Tuning for Recovery](#)
- [Tuning for Refresh](#)
- [Tuning Tuple Mover Pool Settings](#)
- [Tuning for Machine Learning](#)

Restricting Vertica to Take Only 60% of Memory

Scenario

You have a single node application that embeds Vertica, and some portion of the RAM needs to be devoted to the application process. In this scenario, you want to limit Vertica to use only 60% of the available RAM.

Solution

Set the MAXMEMORYSIZE parameter of the GENERAL pool to the desired memory size. See [Resource Pool Architecture](#) for a discussion on resource limits.

Tuning for Recovery

Scenario

You have a large database that contains a single large table with two projections, and with default settings, recovery is taking too long. You want to give recovery more memory to improve speed.

Solution

Set the `PLANNEDCONCURRENCY` and `MAXCONCURRENCY` setting of the recovery pool to 1 so that recovery can take as much memory as possible from the `GENERAL` pool and run only one thread at once.

Note: This setting could slow down other queries in your system.

Tuning for Refresh

Scenario

When a refresh operation is running, system performance is affected and user queries get rejected. You want to reduce the memory usage of the refresh job.

Solution

Set the MEMORYSIZE parameter of the refresh pool to a fixed value. The Resource Manager then tunes the refresh query to only use this amount of memory.

Tip: Remember to reset the refresh pool MEMORYSIZE back to 0% after the refresh operation completes so memory can be used for other operations.

Tuning Tuple Mover Pool Settings

Scenario

During loads, you occasionally notice spikes in the number of ROS containers, and you would like to make the Tuple Mover more aggressive.

Solution

Increase the `MAXCONCURRENCY` parameter of the TM pool to 3 or higher. This setting ensures that the Tuple Mover can run more than one mergeout thread, so if a large mergeout is in progress, smaller ROS containers can also be merged, thus preventing a buildup of ROS containers.

Tuning for Machine Learning

Scenario

You have a large number of machine learning algorithms running, and you want to improve the performance of the queries. In this scenario, you want to give machine learning functions more memory to improve their performance.

Solution

Vertica executes the machine learning functions in the BLOBDATA resource pool. To improve the performance of the machine learning functions increase the MAXMEMORYSIZE parameter of the BLOBDATA pool. If a query using the BLOBDATA pool exceeds its query planning budget, then it spills to disk.

For more information about tuning your query budget, see [Target Memory Determination for Queries in Concurrent Environments](#).

See Also

- [ALTER RESOURCE POOL](#)
- [Managing Resources At Query Run Time](#)
- [Built-In Pool Configuration](#)
- [Built-In Pools](#)

Reducing Query Run Time

The run time of queries depends on the complexity of the query, the number of operators in the plan, data volumes, and projection design. If the system is bottlenecked on either I/O or CPU, queries could run more slowly than expected. In most cases, high CPU usage can be alleviated by better projection design, and high I/O is usually due to contention because of operations like joins and sorts that spill to disk. However, there is no single solution to fix high CPU or high I/O usage, so queries must be examined and tuned individually.

You can evaluate a slow-running query in two ways:

- Prefix the query with [EXPLAIN](#) to view the optimizer's query plan.
- Examine the execution profile by querying the system table [EXECUTION_ENGINE_PROFILES](#)

Examining the query plan often reveals one or more of the following:

- Suboptimal projection sort order
- Predicate evaluation on an unsorted or unencoded column
- Use of `GROUPBY HASH` instead of `GROUPBY PIPE`

See [Creating Custom Designs](#) to understand projection design techniques. The Database Designer automatically applies these techniques to suggest optimal designs for queries.

Profiling

Vertica provides profiling mechanisms that let you determine how well the database is performing. For example, Vertica can collect profiling data for a single statement, a single session, or for all sessions on all nodes. For details, see [Profiling Database Performance](#).

Managing System Resource Usage

You can use the [Using System Tables](#) to track overall resource usage on your cluster. These and the other system tables are described in the [Vertica System Tables](#).

If your queries are experiencing errors due to resource unavailability, you can use the following system tables to obtain more details:

System Table	Description
RESOURCE_REJECTIONS	Monitors requests for resources that are rejected by the Resource Manager.
DISK_RESOURCE_REJECTIONS	Monitors requests for resources that are rejected due to disk space shortages. See Managing Disk Space for more information.

When requests for resources of a certain type are being rejected, do one of the following:

- Increase the resources available on the node by adding more memory, more disk space, and so on. See [Managing Disk Space](#).
- Reduce the demand for the resource by reducing the number of users on the system (see [Managing Sessions](#)), rescheduling operations, and so on.

The `LAST_REJECTED_VALUE` field in `RESOURCE_REJECTIONS` indicates the cause of the problem. For example:

- The message `Usage of a single requests exceeds high limit` means that the system does not have enough of the resource available for the single request. A common example occurs when the file handle limit is set too low and you are loading a table with a large number of columns.
- The message `Timed out or Canceled waiting for resource reservation` usually means that there is too much contention for the resource because the hardware platform cannot support the number of concurrent users using it.

Managing Sessions

Vertica provides several methods for database administrators to view and control sessions. The methods vary according to the type of session:

- External (user) sessions are initiated by vsql or programmatic (ODBC or JDBC) connections and have associated client state.
- Internal (system) sessions are initiated by Vertica and have no client state.

Configuring Maximum Sessions

The maximum number of per-node user sessions is set by the configuration parameter `MaxClientSessions` parameter, by default 50. You can set `MaxClientSessions` parameter to any value between 0 and 1000. In addition to this maximum, Vertica also allows up to five administrative sessions per node.

For example:

```
=> ALTER DATABASE mydb SET MaxClientSessions = 100;
```

Note: If you use the Administration Tools "Connect to Database" option, Vertica will attempt connections to other nodes if a local connection does not succeed. These cases can result in more successful "Connect to Database" commands than you would expect given the `MaxClientSessions` value.

Viewing Sessions

The system table [SESSIONS](#) contains detailed information about user sessions and returns one row per session. Superusers have unrestricted access to all database metadata. Access for other users varies according to their [privileges](#).

Interrupting and Closing Sessions

You can interrupt a running statement with the Vertica function [INTERRUPT_STATEMENT](#). Interrupting a running statement returns a session to an idle state:

- No statements or transactions are running.
- No locks are held.
- The database is doing no work on behalf of the session.

Closing a user session interrupts the session and disposes of all state related to the session, including client socket connections for the target sessions. The following Vertica functions close one or more user sessions:

- [CLOSE_SESSION](#)
- [CLOSE_ALL_SESSIONS](#)
- [CLOSE_USER_SESSIONS](#)
- [SHUTDOWN](#)

SELECT statements that call these functions return after the interrupt or close message is delivered to all nodes. The function might return before Vertica completes execution of the interrupt or close operation. Thus, there might be a delay after the statement returns and the interrupt or close takes effect throughout the cluster. To determine if the session or transaction ended, query the `SESSIONS` system table.

In order to shut down a database, you must first close all user sessions. For more about database shutdown, see [Stopping the Database](#).

Managing Load Streams

You can use the [Using System Tables](#) to keep track of data being loaded on your cluster.

System Table	Description
LOAD_STREAMS	Monitors load metrics for each load stream on each node.

For details about Vertica system tables, see [Vertica System Tables](#).

When a `COPY` statement using the `DIRECT` option is in progress, the `ACCEPTED_ROW_COUNT` value can increase during parsing. This value can reach the maximum number of rows in the input file.

If `COPY` reads input data from multiple named pipes, the `PARSE_COMPLETE_PERCENT` value remains at zero (0) until *all* named pipes return an EOF. While `COPY` awaits an EOF from multiple pipes, it can appear to be hung. However, before canceling the `COPY` statement, check your [system CPU and disk accesses](#) to determine if any activity is in progress.

In a typical load, the `PARSE_COMPLETE_PERCENT` value can either increase slowly or jump quickly to 100%, if you are loading from named pipes or `STDIN`. However, `SORT_COMPLETE_PERCENT` remains at 0 when loading from named pipes or `STDIN`. After `PARSE_COMPLETE_PERCENT` reaches 100%, `SORT_COMPLETE_PERCENT` increases to 100%. Depending on the data sizes, a significant lag can occur between the time `PARSE_COMPLETE_PERCENT` reaches 100% and the time `SORT_COMPLETE_PERCENT` begins to increase.

Monitoring Vertica

This section describes some of the ways in which you can monitor the health of your database.

You can also monitor the Vertica database using MC, as described in [Monitoring Using MC](#).

Monitoring Log Files

When a Database Is Running

When a Vertica database is running, each node in the cluster writes messages into a file named `vertica.log`. For example, the Tuple Mover and the transaction manager write INFO messages into `vertica.log` at specific intervals even when there is no WOS activity.

Note: Vertica often changes the format or content of log files in subsequent releases to benefit both customers and customer support.

To monitor one node in a running database in real time:

1. Log in to the database administrator account on any node in the cluster.
2. In a terminal window enter:

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
```

Note: To monitor your overall database (rather than an individual node/host), use the Data Collector, which records system activities and performance. See [Retaining Monitoring Information](#) for more on Data Collector.

<i>catalog-path</i>	The catalog pathname specified when you created the database. See Creating a Database in the Administrator's Guide.
<i>database-name</i>	The database name (case sensitive)
<i>node-name</i>	The node name, as specified in the database definition. See Viewing a Database in the Administrator's Guide.

When the Database / Node Is Starting up

During system startup, before the vertica log has been initialized to write messages, each node in the cluster writes messages into a file named dbLog. This log is useful to diagnose situations where the database fails to start before it can write messages into vertica.log. The dblog is located at the following path, using catalog-path and database-name as described above:

```
catalog-path/database-name/dbLog
```

See Also

- [Rotating Log Files](#)

Rotating Log Files

Most Linux distributions include the logrotate utility. Using this utility simplifies log file administration. By setting up a logrotate configuration file, you can use the utility to complete one or more of these tasks automatically:

- Compress and rotate log files
- Remove log files automatically
- Email log files to named recipients

You can configure logrotate to complete these tasks at specific intervals, or when log files reach a particular size.

If logrotate is present when Vertica is installed, then Vertica automatically sets this utility to look for configuration files. Thus, logrotate searches for configuration files in the /opt/vertica/config/logrotate directory on each node.

When you create a database, Vertica creates database-specific logrotate configurations on each node in your cluster, which are used by the logrotate utility. It then creates a file with the path /opt/vertica/config/logrotate/<dbname> for each individual database.

For information about additional settings, use the man logrotate command.

Executing the Python script Through the dbadmin logrotate cron Job

During the installation of Vertica, the installer configures a cron job for the dbadmin user. This cron job is configured to execute a Python script that runs the `logrotate` utility. You can view the details of this cron job by viewing the `dbadmin.cron` file, which is located in the `/opt/vertica/config` directory.

If you want to customize a cron job to configure logrotate for your Vertica database, you *must* create the cron job under the dbadmin user.

Using the Administration Tools Logrotate Utility

You can use the `admintools logrotate` option to help configure logrotate scripts for a database and distribute the scripts across the cluster. The `logrotate` option allows you to specify:

- How often to rotate logs
- How large logs can become before being rotated
- How long to keep the logs

Example:

The following example shows you how to set up log rotation on a weekly schedule and keeps for three months (12 logs).

```
$ admintools -t logrotate -d <dbname> -r weekly -k 12
```

See [Writing Administration Tools Scripts](#) for more usage information.

Rotating Logs Manually

To implement a custom log rotation process, follow these steps:

1. Rename or archive the existing *vertica.log* file. For example:

```
$ mv vertica.log vertica.log.1
```

2. Send the Vertica process the USR1 signal, using either of the following approaches:

```
$ killall -USR1 vertica
```

or

```
$ ps -ef | grep -i vertica  
$ kill -USR1 process-id
```

See Also

- [Monitoring Log Files](#)

Monitoring Process Status (ps)

You can use `ps` to monitor the database and Spread processes running on each node in the cluster. For example:

```
$ ps aux | grep /opt/vertica/bin/vertica
$ ps aux | grep /opt/vertica/spread/sbin/spread
```

You should see one Vertica process and one Spread process on each node for common configurations. To monitor Administration Tools and connector processes:

```
$ ps aux | grep vertica
```

There can be many connection processes but only one Administration Tools process.

Monitoring Linux Resource Usage

You should monitor system resource usage on any or all nodes in the cluster. You can use System Activity Reporting (SAR) to monitor resource usage.

Micro Focus recommends that you install `pstack` and `sysstat` to help monitor Linux resources. The `SYSSTAT` package contains utilities for monitoring system performance and usage activity, such as `sar`, as well as tools you can schedule via `cron` to collect performance and activity data. See the [SYSSTAT Web page](#) for details.

The `pstack` utility lets you print a stack trace of a running process. See the [PSTACK man page](#) for details.

1. Log in to the database administrator account on any node.
2. Run the `top` utility

```
$ top
```

A high CPU percentage in `top` indicates that Vertica is CPU-bound. For example:

```
top - 11:44:28 up 53 days, 23:47, 9 users, load average: 0.91, 0.97, 0.81
Tasks: 123 total, 1 running, 122 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.9%us, 1.3%sy, 0.0%ni, 71.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4053136 total, 3882020k used, 171116 free, 407688 buffers
Swap: 4192956 total, 176k used, 4192780 free, 1526436 cached
  PID USER      PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  COMMAND
 13703 dbadmin    1   0 1374m 678m 55m  S 99.9 17.1   6:21.70 vertica
  2606 root       16   0 32152  11m 2508  S  1.0  0.3    0:16.97 X
    1 root       16   0  4748   552  456  S  0.0  0.0    0:01.51 init
    2 root       RT  -5     0     0     0  S  0.0  0.0    0:04.92 migration/0
    3 root       34  19     0     0     0  S  0.0  0.0    0:11.75 ksoftirqd/0
...
```

Some possible reasons for high CPU usage are:

- The Tuple Mover runs automatically and thus consumes CPU time even if there are no connections to the database.
- Log in as root and change the Linux parameter `swappiness` to 1.

```
# echo 1 > /proc/sys/vm/swappiness
```

Setting swappiness in this manner does not save the value after a reboot. To permanently set the swappiness value, add or update the following in `/etc/sysctl.conf`:

```
vm.swappiness = 1
```

You can also check the swappiness value as follows:

```
cat /proc/sys/vm/swappiness
```

■ Some information sources:

- [Red Hat](#)
- [Indiana University Unix Systems Support Group](#)

3. Run the `iostat` utility. A high idle time in `top` at the same time as a high rate of blocks read in `iostat` indicates that Vertica is disk-bound. For example:

```
$ /usr/bin/iostat
Linux 2.6.18-164.el5 (qa01)      02/05/2011
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.77    2.32   0.76    0.68    0.00   95.47

Device:            tps    Blk_read/s  Blk_wrtn/s  Blk_read  Blk_wrtn
hda                 0.37         3.40         10.37     2117723   6464640
sda                 0.46         1.94         18.96     1208130   11816472
sdb                 0.26         1.79         15.69     1114792   9781840
sdc                 0.24         1.80         16.06     1119304   10010328
sdd                 0.22         1.79         15.52     1117472   9676200
md0                 8.37         7.31         66.23     4554834   41284840
```

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

System table	Description
DISK_STORAGE	Monitors the amount of disk storage used by the database on each node.
COLUMN_STORAGE	Monitors the amount of disk storage used by each column of each projection on each node.
PROJECTION_STORAGE	Monitors the amount of disk storage used by each projection on each

System table	Description
	node.

Monitoring Database Size for License Compliance

Your Vertica license can include a data storage allowance. The allowance can consist of data in columnar tables, flex tables, or both types of data. The `AUDIT()` function estimates the columnar table data size and any flex table materialized columns. The `AUDIT_FLEX()` function estimates the amount of `__raw__` column data in flex or columnar tables. In regards to license data limits, data in `__raw__` columns is calculated at 1/10th the size of structured data. Monitoring data sizes for columnar and flex tables lets you plan either to schedule deleting old data to keep your database in compliance with your license, or to consider a license upgrade for additional data storage.

Note: An audit of columnar data includes flex table real and materialized columns, but not `__raw__` column data.

Viewing Your License Compliance Status

Vertica periodically runs an audit of the columnar data size to verify that your database is compliant with your license terms. You can view the results of the most recent audit by calling the `GET_COMPLIANCE_STATUS` function.

```
=> select GET_COMPLIANCE_STATUS();
          GET_COMPLIANCE_STATUS
-----
Raw Data Size: 2.00GB +/- 0.003GB
License Size : 4.000GB
Utilization  : 50%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

Periodically running `GET_COMPLIANCE_STATUS` to monitor your database's license status is usually enough to ensure that your database remains compliant with your license. If your database begins to near its columnar data allowance, you can use the other auditing functions

described below to determine where your database is growing and how recent deletes affect the database size.

Manually Auditing Columnar Data Usage

You can manually check license compliance for all columnar data in your database using the [AUDIT_LICENSE_SIZE](#) function. This function performs the same audit that Vertica periodically performs automatically. The [AUDIT_LICENSE_SIZE](#) check runs in the background, so the function returns immediately. You can then query the results using [GET_COMPLIANCE_STATUS](#).

Note: When you audit columnar data, the results include any flex table real and materialized columns, but not data in the `__raw__` column. Materialized columns are virtual columns that you have promoted to real columns. Columns that you define when creating a flex table, or which you add with `ALTER TABLE . . . ADD COLUMN` statements are real columns. All `__raw__` columns are real columns. However, since they consist of unstructured or semi-structured data, they are audited separately.

An alternative to `AUDIT_LICENSE_SIZE` is to use the [AUDIT](#) function to audit the size of the columnar tables in your entire database by passing an empty string to the function. This function operates synchronously, returning when it has estimated the size of the database.

```
=> SELECT AUDIT('');  
  AUDIT  
-----  
  76376696  
(1 row)
```

The size of the database is reported in bytes. The `AUDIT` function also allows you to control the accuracy of the estimated database size using additional parameters. See the entry for the [AUDIT](#) function in the SQL Reference Manual for full details. Vertica does not count the `AUDIT` function results as an official audit. It takes no license compliance actions based on the results.

Note: The results of the `AUDIT` function do not include flex table data in `__raw__` columns. Use the [AUDIT_FLEX](#) function to monitor data usage flex tables.

Manually Auditing `__raw__` Column Data

You can use the [AUDIT_FLEX](#) function to manually audit data usage for flex or columnar tables with a `__raw__` column. The function calculates the encoded, compressed data stored in ROS containers for any `__raw__` columns. Materialized columns in flex tables are calculated by the

AUDIT function. The [AUDIT_FLEX](#) results do not include data in the `__raw__` columns of temporary flex tables.

Targeted Auditing

If audits determine that the columnar table estimates are unexpectedly large, consider schemas, tables, or partitions that are using the most storage. You can use the AUDIT function to perform targeted audits of schemas, tables, or partitions by supplying the name of the entity whose size you want to find. For example, to find the size of the `online_sales` schema in the [VMart](#) example database, run the following command:

```
=> SELECT AUDIT('online_sales');  
  AUDIT  
-----  
 35716504  
(1 row)
```

You can also change the granularity of an audit to report the size of each object in a larger entity (for example, each table in a schema) by using the `granularity` argument of the AUDIT function. See the [AUDIT](#) function in the SQL Reference Manual.

Using Management Console to Monitor License Compliance

You can also get information about data storage of columnar data (for columnar tables and for materialized columns in flex tables) through the Management Console. This information is available in the database Overview page, which displays a grid view of the database's overall health.

- The needle in the license meter adjusts to reflect the amount used in megabytes.
- The grace period represents the term portion of the license.
- The Audit button returns the same information as the `AUDIT()` function in a graphical representation.
- The Details link within the License grid (next to the Audit button) provides historical information about license usage. This page also shows a progress meter of percent used toward your license limit.

Monitoring Elastic Cluster Rebalancing

Vertica includes system tables that can be used to monitor the rebalance status of an elastic cluster and gain general insight to the status of elastic cluster on your nodes.

- The [REBALANCE_TABLE_STATUS](#) table provides general information about a rebalance. It shows, for each table, the amount of data that has been separated, the amount that is currently being separated, and the amount to be separated. It also shows the amount of data transferred, the amount that is currently being transferred, and the remaining amount to be transferred (or an estimate if storage is not separated).

Note: If multiple rebalance methods were used for a single table (for example, the table has unsegmented and segmented projections), the table may appear multiple times - once for each rebalance method.

- [REBALANCE_PROJECTION_STATUS](#) can be used to gain more insight into the details for a particular projection that is being rebalanced. It provides the same type of information as above, but in terms of a projection instead of a table.

In each table, *separated_percent* and *transferred_percent* can be used to determine overall progress.

Historical Rebalance Information

Historical information about work completed is retained, so use the predicate "*where is_latest*" to restrict the output to only the most recent or current rebalance activity. The historical data may include information about dropped projections or tables. If a table or projection has been dropped and information about the anchor table is not available, then NULL is displayed for the *table_id* and "<unknown>" is displayed for the *table_name*. Information on dropped tables is still useful, for example, in providing justification for the duration of a task.

Monitoring Parameters

The following table describes the monitoring parameters for configuring Vertica.

Parameters	Description
SnmpTrapDestinationsList	<p>Defines where Vertica sends traps for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: none</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'localhost 162 public';</pre>
SnmpTrapsEnabled	<p>Enables event trapping for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SnmpTrapsEnabled = 1;</pre>
SnmpTrapEvents	<p>Define which events Vertica traps through SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: Low Disk Space, Read Only File System, Loss of K Safety, Current Fault Tolerance at Critical Level, Too Many ROS Containers, WOS Over Flow, Node State Change, Recovery Failure, Stale Checkpoint, and CRC Mismatch.</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SnmpTrapEvents = 'Low Disk Space, Recovery Failure';</pre>
SyslogEnabled	<p>Enables event trapping for syslog. See Configuring Reporting for Syslog.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SyslogEnabled = 1);</pre>
SyslogEvents	<p>Defines events that generate a syslog entry. See Configuring Reporting for Syslog.</p> <p>Default Value: none</p> <p>Example:</p> <pre>ALTER DATABASE mydb SET SyslogEvents = 'Low Disk Space, Recovery Failure';</pre>
SyslogFacility	<p>Defines which SyslogFacility Vertica uses. See Configuring</p>

Parameters	Description
	<p data-bbox="576 283 844 315">Reporting for Syslog.</p> <p data-bbox="576 336 836 367">Default Value: user</p> <p data-bbox="576 388 706 420">Example:</p> <pre data-bbox="576 441 1128 472">ALTER DATABASE mydb SET SyslogFacility = 'ftp';</pre>

Monitoring Events

To help you monitor your database system, Vertica traps and logs significant events that affect database performance and functionality if you do not address their root causes. This section describes where events are logged, the types of events that Vertica logs, how to respond to these events, the information that Vertica provides for these events, and how to configure event monitoring.

Event Logging Mechanisms

Vertica posts events to the following mechanisms:

Mechanism	Description
vertica.log	All events are automatically posted to vertica.log. See Monitoring the Log Files .
ACTIVE_EVENTS	This SQL system table provides information about all open events. See Using System Tables and ACTIVE_EVENTS .
SNMP	To post traps to SNMP, enable global reporting in addition to each individual event you want trapped. See Configuring Event Reporting .
Syslog	To log events to syslog, enable event reporting for each individual event you want logged. See Configuring Event Reporting .

Event Severity Types

Event names are sensitive to case and spaces. Vertica logs the following events:

Event Name	Event Type	Description	Action
Low Disk Space	0	The database is running out of disk space or a disk is failing or there is a I/O hardware failure.	It is imperative that you add more disk space or replace the failing disk or hardware as soon as possible.

Event Name	Event Type	Description	Action
			<p>Check <code>dmesg</code> to see what caused the problem.</p> <p>Also, use the DISK_RESOURCE_REJECTIONS system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected. See Managing Disk Space for more information about low disk space.</p>
Read Only File System	1	The database does not have write access to the file system for the data or catalog paths. This can sometimes occur if Linux remounts a drive due to a kernel issue.	Modify the privileges on the file system to give the database write access.
Loss Of K Safety	2	<p>The database is no longer K-Safe because there are insufficient nodes functioning within the cluster. Loss of K-safety causes the database to shut down.</p> <p>In a four-node cluster, for example, $K\text{-safety}=1$. If one node fails, the fault tolerance is at a critical level. If two nodes fail, the system loses K-safety.</p>	If a system shuts down due to loss of K-safety, you need to recover the system. See Failure Recovery in the Administrator's Guide.
Current Fault Tolerance at Critical Level	3	One or more nodes in the cluster have failed. If the database loses one more node, it is no longer K-Safe and it shuts down. (For	Restore any nodes that have failed or been shut down.

Event Name	Event Type	Description	Action
		example, a four-node cluster is no longer K-safe if two nodes fail.)	
Too Many ROS Containers	4	Due to heavy data load conditions, there are too many ROS containers. This occurs when the Tuple Mover falls behind in performing mergeout operations. The resulting excess number of ROS containers can exhaust all available system resources. To prevent this, Vertica automatically rolls back all transactions that would load data until the Tuple Mover has time to catch up.	<p>You might need to adjust the Tuple Mover's configuration parameters to compensate for the load pattern or rate. See Managing the Tuple Mover in the Administrator's Guide for details.</p> <p>You can query the TUPLE_MOVER_OPERATIONS table to monitor mergeout activity. However, the Tuple Mover does not immediately start a mergeout when a projection reaches the limit of ROS containers, so you may not see a mergeout in progress when receiving this error.</p> <p>If waiting for a mergeout does not resolve the error, the problem probably is related to insufficient RAM.. A good rule of thumb is that system RAM in MB divided by 6 times the number of columns in the largest table should be greater than 10. For example, for a 100 column table you would want at least 6GB of RAM ($6144\text{MB} / (6 * 100) = 10.24$) to handle continuous loads.</p>
WOS Over Flow	5	The WOS cannot hold all the data that you are attempting to load. This means that the	Consider loading the data to disk (ROS) instead of memory (WOS) or splitting the fact

Event Name	Event Type	Description	Action
		copy fails and the transaction rolls back.	table load file into multiple pieces and then performing multiple loads in sequence. You might also consider making the Tuple Mover's moveout operation more aggressive. See Managing the Tuple Mover in Administrator's Guide.
Node State Change	6	The node state has changed.	Check the status of the node.
Recovery Failure	7	The database was not restored to a functional state after a hardware or software related failure.	The reason for recovery failure can vary. See the event description for more information about your specific situation.
Recovery Error	8	The database encountered an error while attempting to recover. If the number of recovery errors exceeds Max Tries, the Recovery Failure event is triggered. See Recovery Failure within this table.	The reason for a recovery error can vary. See the event description for more information about your specific situation.
Recovery Lock Error	9	A recovering node could not obtain an S lock on the table. If you have a continuous stream of COPY commands in progress, recovery might not be able to obtain this lock even after multiple re-tries.	Either momentarily stop the loads or pick a time when the cluster is not busy to restart the node and let recovery proceed.
Recovery Projection Retrieval Error	10	Vertica was unable to retrieve information about a projection.	The reason for a recovery projection retrieval error can vary. See the event description

Event Name	Event Type	Description	Action
			for more information about your specific situation.
Refresh Error	11	The database encountered an error while attempting to refresh.	The reason for a refresh error can vary. See the event description for more information about your specific situation.
Refresh Lock Error	12	The database encountered a locking error during refresh.	The reason for a refresh error can vary. See the event description for more information about your specific situation.
Tuple Mover Error	13	The database encountered an error while attempting to move the contents of the Write Optimized Store (WOS) into the Read Optimized Store (ROS).	The reason for a Tuple Mover error can vary. See the event description for more information about your specific situation.
Timer Service Task Error	14	An error occurred in an internal scheduled task.	Internal use only
Stale Checkpoint	15	Data in the WOS has not been completely moved out in a timely manner. An UNSAFE shutdown could require reloading a significant amount of data.	Be sure that Moveout operations are executing successfully. Check the <code>vertica.log</code> files for errors related to Moveout.
CRC Mismatch	16	The Cyclic Redundancy Check returned an error or errors while fetching data.	Review the <code>vertica.log</code> file or the SNMP trap utility to review the errors. For more information see Handling CRC Errors .

Event Data

To help you interpret and solve the issue that triggered an event, each event provides a variety of data, depending upon the event logging mechanism used.

The following table describes the event data and indicates where it is used.

vertica.log	ACTIVE_EVENTS (column names)	SNMP	Syslog	Description
N/A	NODE_NAME	N/A	N/A	The node where the event occurred.
Event Code	EVENT_CODE	Event Type	Event Code	A numeric ID that indicates the type of event. See Event Types in the previous table for a list of event type codes.
Event Id	EVENT_ID	Event OID	Event Id	A unique numeric ID that identifies the specific event.
Event Severity	EVENT_SEVERITY	Event Severity	Event Severity	The severity of the event from highest to lowest. These

vertica.log	ACTIVE_EVENTS (column names)	SNMP	Syslog	Description
				<p>events are based on standard syslog severity types:</p> <ul style="list-style-type: none"> 0 - Emergency 1 - Alert 2 - Critical 3 - Error 4 - Warning 5 - Notice 6 - Info 7 - Debug
PostedTimestamp	EVENT_POSTED_TIMESTAMP	N/A	PostedTimestamp	<p>The year, month, day, and time the event was reported. Time is provided as military time.</p>
ExpirationTimestamp	EVENT_EXPIRATION	N/A	ExpirationTimestamp	<p>The time at which this event expires. If the same</p>

vertica.log	ACTIVE_ EVENTS (column names)	SNMP	Syslog	Description
				event is posted again prior to its expiration time, this field gets updated to a new expiration time.
EventCodeDescription	EVENT_ CODE_ DESCRIPTION	Description	EventCodeDescription	A brief description of the event and details pertinent to the specific situation.
ProblemDescription	EVENT_ PROBLEM_ DESCRIPTION	Event Short Description	ProblemDescription	A generic description of the event.
N/A	REPORTING_ NODE	Node Name	N/A	The name of the node within the cluster that reported the event.
DatabaseName	N/A	Database Name	DatabaseName	The name of the database that is impacted by the event.
N/A	N/A	Host Name	Hostname	The name of

<code>vertica.log</code>	<code>ACTIVE_EVENTS</code> (column names)	SNMP	Syslog	Description
				the host within the cluster that reported the event.
N/A	N/A	Event Status	N/A	The status of the event. It can be either: 1 - Open 2 - Clear

Configuring Event Reporting

Event reporting is automatically configured for `vertica.log`, and current events are automatically posted to the `ACTIVE_EVENTS` system table. You can also configure Vertica to post events to `syslog` and `SNMP`.

Configuring Reporting for Syslog

Syslog is a network-logging utility that issues, stores, and processes meaningful log messages. It is designed so DBAs can keep machines up and running, and it is a useful way to get heterogeneous data into a single data repository.

To log events to syslog, enable event reporting for each individual event you want logged. Messages are logged, by default, in `/var/log/messages`.

Configuring event reporting to syslog consists of:

1. Enabling Vertica to trap events for syslog.
2. Defining which events Vertica traps for syslog.

Micro Focus strongly suggests that you trap the Stale Checkpoint event.

3. Defining which syslog facility to use.

Enabling Vertica to Trap Events for Syslog

To enable event trapping for syslog, issue the following SQL command:

```
=> ALTER DATABASE mydb SET SyslogEnabled = 1;
```

To disable event trapping for syslog, issue the following SQL command:

```
=> ALTER DATABASE mydb SET SyslogEnabled = 0;
```

Defining Events to Trap for Syslog

To define events that generate a syslog entry, issue the following SQL command, where `Event_Name` is one of the events described in the list below the command:

```
=> ALTER DATABASE mydb SET SyslogEvents = 'Event_Name, Event_Name';
```

- Low Disk Space
- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error

- Refresh Lock Error
- Tuple Mover Error
- Timer Service Task Error
- Stale Checkpoint

The following example generates a syslog entry for low disk space and recovery failure:

```
=> ALTER DATABASE mydb SET SyslogEvents = 'Low Disk Space, Recovery Failure';
```

Defining the SyslogFacility to Use for Reporting

The syslog mechanism allows for several different general classifications of logging messages, called facilities. Typically, all authentication-related messages are logged with the `auth` (or `authpriv`) facility. These messages are intended to be secure and hidden from unauthorized eyes. Normal operational messages are logged with the `daemon` facility, which is the collector that receives and optionally stores messages.

The `SyslogFacility` directive allows all logging messages to be directed to a different facility than the default. When the directive is used, *all* logging is done using the specified facility, both authentication (secure) and otherwise.

To define which `SyslogFacility` Vertica uses, issue the following SQL command:

```
=> ALTER DATABASE mydb SET SyslogFacility = 'Facility_Name';
```

Where the facility-level argument `<Facility_Name>` is one of the following:

<ul style="list-style-type: none">• <code>auth</code>	<ul style="list-style-type: none">• <code>uucp</code> (UUCP subsystem)
<ul style="list-style-type: none">• <code>authpriv</code> (Linux only)	<ul style="list-style-type: none">• <code>local0</code> (local use 0)
<ul style="list-style-type: none">• <code>cron</code>	<ul style="list-style-type: none">• <code>local1</code> (local use 1)
<ul style="list-style-type: none">• <code>daemon</code>	<ul style="list-style-type: none">• <code>local2</code> (local use 2)
<ul style="list-style-type: none">• <code>ftp</code> (Linux only)	<ul style="list-style-type: none">• <code>local3</code> (local use 3)
<ul style="list-style-type: none">• <code>lpr</code> (line printer subsystem)	<ul style="list-style-type: none">• <code>local4</code> (local use 4)
<ul style="list-style-type: none">• <code>mail</code> (mail system)	<ul style="list-style-type: none">• <code>local5</code> (local use 5)

<ul style="list-style-type: none">• news (network news subsystem)	<ul style="list-style-type: none">• local6 (local use 6)
<ul style="list-style-type: none">• user (default system)	<ul style="list-style-type: none">• local7 (local use 7)

See Also

- [Event Reporting Examples](#)
- [Configuration Parameters](#)

Configuring Reporting for SNMP

Configuring event reporting for SNMP consists of:

1. Configuring Vertica to enable event trapping for SNMP as described below.
2. Importing the Vertica Management Information Base (MIB) file into the SNMP monitoring device.

The Vertica MIB file allows the SNMP trap receiver to understand the traps it receives from Vertica. This, in turn, allows you to configure the actions it takes when it receives traps.

Vertica supports the SNMP V1 trap protocol, and it is located in `/opt/vertica/sbin/VERTICA-MIB`. See the documentation for your SNMP monitoring device for more information about importing MIB files.

3. Configuring the SNMP trap receiver to handle traps from Vertica.

SNMP trap receiver configuration differs greatly from vendor to vendor. As such, the directions presented here for configuring the SNMP trap receiver to handle traps from Vertica are generic.

Vertica traps are single, generic traps that contain several fields of identifying information. These fields equate to the event data described in [Monitoring Events](#). However, the format used for the field names differs slightly. Under SNMP, the field names contain no spaces. Also, field names are pre-pended with “vert”. For example, Event Severity becomes `vertEventSeverity`.

When configuring your trap receiver, be sure to use the same hostname, port, and community string you used to configure event trapping in Vertica.

Examples of network management providers:

- IBM Tivoli
- AdventNet
- Net-SNMP (Open Source)
- Nagios (Open Source)
- Open NMS (Open Source)

See Also

- [Configuration Parameters](#)

Configuring Event Trapping for SNMP

The following events are trapped by default when you configure Vertica to trap events for SNMP:

- Low Disk Space
- Read Only File System
- Loss of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Stale Checkpoint
- CRC Mismatch

To Configure Vertica to Trap Events for SNMP

1. Enable Vertica to trap events for SNMP.
2. Define where Vertica sends the traps.
3. Optionally redefine which SNMP events Vertica traps.

Note: After you complete steps 1 and 2 above, Vertica automatically traps the default SNMP events. Only perform step 3 if you want to redefine which SNMP events are trapped. Micro Focus recommends that you trap the `Stale Checkpoint` event even if you decide to reduce the number events Vertica traps for SNMP. The specific settings you define have no effect on traps sent to the log. All events are trapped to the log.

To Enable Event Trapping for SNMP

Use the following SQL command:

```
=> ALTER DATABASE mydb SET SnmpTrapsEnabled = 1;
```

To Define Where Vertica Send Traps

Use the following SQL command, where `Host_name` and `port` identify the computer where SNMP resides, and `CommunityString` acts like a password to control Vertica's access to the server:

```
=> ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'host_name port CommunityString';
```

For example:

```
=> ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'localhost 162 public';
```

You can also specify multiple destinations by specifying a list of destinations, separated by commas:

```
=> ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'host_name1 port1 CommunityString1, hostname2 port2 CommunityString2';
```

Note: : Setting multiple destinations sends any SNMP trap notification to all destinations listed.

To Define Which Events Vertica Traps

Use the following SQL command, where `Event_Name` is one of the events in the list below the command:

```
=> ALTER DATABASE mydb SET SnmpTrapEvents = 'Event_Name1, Even_Name2';
```

- Low Disk Space
- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error
- Tuple Mover Error
- Stale Checkpoint
- CRC Mismatch

Note: The above values are case sensitive.

The following example specifies two event names:

```
=> ALTER DATABASE mydb SET SnmpTrapEvents = 'Low Disk Space, Recovery Failure';
```

Verifying SNMP Configuration

To create a set of test events that checks SNMP configuration:

1. Set up SNMP trap handlers to catch Vertica events.
2. Test your setup with the following command:

```
SELECT SNMP_TRAP_TEST();
       SNMP_TRAP_TEST
-----
Completed SNMP Trap Test
(1 row)
```

Event Reporting Examples

Vertica.log

The following example illustrates a Too Many ROS Containers event posted and cleared within vertica.log:

```
08/14/15 15:07:59 thr:nameless:0x45a08940 [INFO] Event Posted: Event Code:4 Event Id:0 Event
Severity: Warning [4] PostedTimestamp:
2015-08-14 15:07:59.253729 ExpirationTimestamp: 2015-08-14 15:08:29.253729
EventCodeDescription: Too Many ROS Containers ProblemDescription:
Too many ROS containers exist on this node. DatabaseName: TESTDB
Hostname: fc6-1.example.com
08/14/15 15:08:54 thr:Ageout Events:0x2aaab0015e70 [INFO] Event Cleared:
Event Code:4 Event Id:0 Event Severity: Warning [4] PostedTimestamp:
2015-08-14 15:07:59.253729 ExpirationTimestamp: 2015-08-14 15:08:53.012669
EventCodeDescription: Too Many ROS Containers ProblemDescription:
Too many ROS containers exist on this node. DatabaseName: TESTDB
Hostname: fc6-1.example.com
```

SNMP

The following example illustrates a Too Many ROS Containers event posted to SNMP:

```
Version: 1, type: TRAPREQUESTEnterprise OID: .1.3.6.1.4.1.31207.2.0.1
Trap agent: 72.0.0.0
Generic trap: ENTERPRISESPECIFIC (6)
Specific trap: 0
.1.3.6.1.4.1.31207.1.1 ---> 4
```

```
.1.3.6.1.4.1.31207.1.2 ---> 0  
.1.3.6.1.4.1.31207.1.3 ---> 2008-08-14 11:30:26.121292  
.1.3.6.1.4.1.31207.1.4 ---> 4  
.1.3.6.1.4.1.31207.1.5 ---> 1  
.1.3.6.1.4.1.31207.1.6 ---> site01  
.1.3.6.1.4.1.31207.1.7 ---> suse10-1  
.1.3.6.1.4.1.31207.1.8 ---> Too many ROS containers exist on this node.  
.1.3.6.1.4.1.31207.1.9 ---> QATESTDB  
.1.3.6.1.4.1.31207.1.10 ---> Too Many ROS Containers
```

Syslog

The following example illustrates a Too Many ROS Containers event posted and cleared within syslog:

```
Aug 14 15:07:59 fc6-1 vertica: Event Posted: Event Code:4 Event Id:0 Event Severity: Warning [4]  
PostedTimestamp: 2015-08-14 15:07:59.253729 ExpirationTimestamp:  
2015-08-14 15:08:29.253729 EventCodeDescription: Too Many ROS Containers ProblemDescription:  
Too many ROS containers exist on this node. DatabaseName: TESTDB Hostname: fc6-1.example.com  
Aug 14 15:08:54 fc6-1 vertica: Event Cleared: Event Code:4 Event Id:0 Event Severity:  
Warning [4] PostedTimestamp: 2015-08-14 15:07:59.253729 ExpirationTimestamp:  
2015-08-14 15:08:53.012669 EventCodeDescription: Too Many ROS Containers ProblemDescription:  
Too many ROS containers exist on this node. DatabaseName: TESTDB Hostname: fc6-1.example.com
```

Using System Tables

Vertica provides an API (application programming interface) for monitoring various features and functions within a database in the form of system tables. These tables provide a robust, stable set of views that let you monitor information about your system's resources, background processes, workload, and performance, allowing you to more efficiently profile, diagnose, and view historical data equivalent to load streams, query profiles, tuple mover operations, and more. Because Vertica collects and retains this information automatically, you don't have to manually set anything.

You can write queries against system tables with full SELECT support the same way you perform query operations on base and temporary tables. You can query system tables using expressions, predicates, aggregates, analytics, subqueries, and joins. You can also save system table query results into a user table for future analysis. For example, the following query creates a table, `mynode`, selecting three node-related columns from the `V_CATALOG.NODES` system table:

```
VMart=> CREATE TABLE mynode AS SELECT node_name, node_state, node_address
FROM nodes;
CREATE TABLE
VMart=> SELECT * FROM mynode;
   node_name      | node_state | node_address
-----+-----+-----
v_vmart_node0001 | UP         | 192.168.223.11
(1 row)
```

Note: You cannot query system tables if the database cluster is in a recovering state. The database refuses connection requests and cannot be monitored. Vertica also does not support DDL and DML operations on system tables.

Where System Tables Reside

System tables are grouped into the following schemas:

- `V_CATALOG` – information about persistent objects in the catalog
- `V_MONITOR` – information about transient system state

These schemas reside in the default search path so there is no need to specify `schema.table` in your queries unless you [change the search path](#) to exclude `V_MONITOR` or `V_CATALOG` or both.

The system tables that make up the monitoring API are described fully in the [SQL Reference Manual](#). You can also use the following command to view all the system tables and their schema:

```
SELECT * FROM system_tables ORDER BY table_schema, table_name;
```

How System Tables Are Organized

Most of the tables are grouped into the following areas:

- System information
- System resources
- Background processes
- Workload and performance

Vertica reserves some memory to help monitor busy systems. Using simple system table queries makes it easier to troubleshoot issues. See also `SYSQUERY` and `SYSDATA` pools under [Built-in pools](#) topic in SQL Reference Manual.

Note: You can use external monitoring tools or scripts to query the system tables and act upon the information, as necessary. For example, when a host failure causes the K-safety level to fall below the desired level, the tool or script can notify the database administrator and/or appropriate IT personnel of the change, typically in the form of an e-mail.

Querying Case-Sensitive Data in System Tables

Some system table data might be stored in mixed case. For example, Vertica stores mixed-case [identifier](#) names the way you specify them in the `CREATE` statement, even though case is ignored when you reference them in queries. When these object names appear as data in the system tables, you'll encounter errors if you use the equality (`=`) predicate because the case must match the stored identifier. In particular, `V_CATALOG.TABLES.TABLE_SCHEMA` and `V_CATALOG.TABLES.TABLE_NAME` columns are case sensitive with equality predicates.

If you don't know how the identifiers are stored, use the case-insensitive operator [ILIKE](#) instead of equality predicates.

For example, given the following schema:

```
=> CREATE SCHEMA SS;  
=> CREATE TABLE SS.TT (c1 int);  
=> CREATE PROJECTION SS.TTP1 AS SELECT * FROM ss.tt UNSEGMENTED ALL NODES;  
=> INSERT INTO ss.tt VALUES (1);
```

If you run a query using the = predicate, Vertica returns 0 rows:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ='ss';  
table_schema | table_name  
-----+-----  
(0 rows)
```

Using the case-insensitive ILIKE predicate returns the expected results:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'ss';  
table_schema | table_name  
-----+-----  
SS           | TT  
(1 row)
```

Examples

The following query uses the VMart example database (see [Introducing the VMart Example Database](#)) to obtain the number of rows and size occupied by each table in the database.

```
=> SELECT t.table_name AS table_name,  
       SUM(ps.wos_row_count + ps.ros_row_count) AS row_count,  
       SUM(ps.wos_used_bytes + ps.ros_used_bytes) AS byte_count  
FROM tables t  
JOIN projections p ON t.table_id = p.anchor_table_id  
JOIN projection_storage ps on p.projection_name = ps.projection_name  
WHERE (ps.wos_used_bytes + ps.ros_used_bytes) > 500000  
GROUP BY t.table_name  
ORDER BY byte_count DESC;  
table_name | row_count | byte_count  
-----+-----+-----  
online_sales_fact | 5000000 | 171987438  
store_sales_fact | 5000000 | 108844666  
store_orders_fact | 300000 | 9240800  
product_dimension | 60000 | 2327964  
customer_dimension | 50000 | 2165897  
inventory_fact | 300000 | 2045900  
(6 rows)
```

The rest of the examples illustrate simple ways to use system tables in queries.

```
=> SELECT table_name FROM columns WHERE data_type ILIKE 'Numeric' GROUP BY table_name;  
table_name  
-----  
n1  
(1 row)  
=> SELECT current_epoch, designed_fault_tolerance, current_fault_tolerance
```

```
FROM SYSTEM;
  current_epoch | designed_fault_tolerance | current_fault_tolerance
-----+-----+-----
          492 |                1 |                1
(1 row)
=> SELECT node_name, total_user_session_count, executed_query_count FROM
query_metrics;
  node_name          | total_user_session_count | executed_query_count
-----+-----+-----
v_vmart_node0001    |                53 |                42
v_vmart_node0002    |                53 |                 0
v_vmart_node0003    |                42 |               120
v_vmart_node0004    |                53 |                 0
(4 rows)
=> SELECT table_schema FROM primary_keys;
table_schema
-----
public
store
online_sales
online_sales
(12 rows)
```

Retaining Monitoring Information

When you query a Vertica system table (described in [Using System Tables](#)), you can get information about currently running queries, the state of various components, and other run-time information. During query execution, Vertica examines the current state of the system and returns information in the result set.

Data Collector

Vertica also provides a utility called the Data Collector (DC), which collects and retains history of important system activities and records essential performance and resource utilization counters.

Data Collector extends system table functionality by:

- Providing a framework for recording events
- Making the information available in system tables
- Requiring few configuration parameter tweaks
- Having negligible impact on performance

You can use the information the Data Collector retains to query the past state of system tables and extract aggregate information, as well as do the following:

- See what actions users have taken
- Locate performance bottlenecks
- Identify potential improvements to Vertica configuration

DC does not collect data for nodes that are down, so no historical data would be available for that node.

Data Collector works in conjunction with the Workload Analyzer (WLA), an advisor tool that intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history. See [Analyzing Workloads](#) for more information about WLA.

Where Is DC Information retained?

Collected data is stored on disk in the DataCollector directory under the Vertica /catalog path. This directory also contains instructions on how to load the monitoring data into another Vertica database. See [Working with Data Collection Logs](#) for details.

DC retains the data it gathers based on retention policies, which a superuser can configure. See [Configuring Data Retention Policies](#).

Data Collector is on by default, but a superuser can disable it if performance issues arise. See [Data Collector Parameters](#) and [Enabling and Disabling Data Collector](#).

DC Tables

Caution: Data Collector tables (prefixed by dc_) reside in the V_INTERNAL schema and are provided for informational purposes only. They are provided as-is and are subject to removal or change without notice. If you use Data Collector tables in scripts or monitoring tools, you might need to change your scripts and tools after a Vertica upgrade. Micro Focus recommends that you use the [Workload Analyzer](#) instead of accessing the Data Collector tables directly.

See Also

- [Data Collector Functions](#)
- [DATA_COLLECTOR](#)
- [ANALYZE_WORKLOAD](#)
- [TUNING_RECOMMENDATIONS](#)

Enabling and Disabling Data Collector

Data Collector is on by default and retains information for all sessions. If performance issues arise, a superuser can disable Data Collector at any time.

To disable the Data Collector:

```
=> ALTER DATABASE mydb SET EnableDataCollector = 0;
```

To re-enable the Data Collector:

```
=> ALTER DATABASE mydb SET EnableDataCollector = 1;
```

See Also

- [Data Collector Parameters](#)

Viewing Current Data Retention Policy

To view the current retention policy for a Data Collector component, use the [GET_DATA_COLLECTOR_POLICY\(\)](#) function and supply the component name as the function's argument.

To retrieve a list of all current component names, query the [V_MONITOR.DATA_COLLECTOR](#) system table, which returns Data Collector components, their current retention policies, and statistics about how much data is retained. For example:

```
mddb=> \xExpanded display is on.
mddb=> SELECT * from data_collector;
-[ RECORD 1 ]-----+-----
node_name          | v_mddb_node0001
component          | AllocationPoolStatistics
table_name         | dc_allocation_pool_statistics
description        | Information about global memory pools, which generally cannot be recovered
without restart
access_restricted | t
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
disk_size_kb       | 256
record_too_big_errors | 0
lost_buffers       | 0
lost_records       | 0
retired_files      | 1429
retired_records    | 647358
current_memory_records | 0
current_disk_records | 1493
current_memory_bytes | 0
current_disk_bytes  | 215737
first_time         | 2012-11-30 07:04:30.000726-05
last_time          | 2012-11-30 07:16:56.000631-05
kb_per_day         | 24377.3198211312
-[ RECORD 2 ]-----+-----
```

The following command returns the retention policy for a specific component, NodeState.

```
=> SELECT get_data_collector_policy('NodeState');
```

The results let you know that 10KB is retained in memory and 100KB on disk:

```
get_data_collector_policy-----  
10KB kept in memory, 100KB kept on disk.  
(1 row)
```

Configuring Data Retention Policies

Data Collector retention policies hold the following information:

- Which component to monitor
- How much memory to retain
- How much disk space to retain

A superuser can modify policies, such as change the amount of data to retain, by invoking the [SET_DATA_COLLECTOR_POLICY\(\)](#) function, as follows:

```
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB' )
```

The `SET_DATA_COLLECTOR_POLICY()` function sets the retention policy for the specified component on all nodes, and lets you specify memory and disk size to retain in kilobytes. Failed nodes receive the new setting when they rejoin the cluster.

For example, the following statement specifies that the `NodeState` component be allocated 50KB of memory and 250KB of disk space:

```
=> SELECT SET_DATA_COLLECTOR_POLICY('NodeState', '50', '250');  
SET_DATA_COLLECTOR_POLICY  
-----  
SET  
(1 row)
```

Before you change a retention policy, you can view the current setting by calling the `GET_DATA_COLLECTOR_POLICY()` function.

You can also use the `GET_DATA_COLLECTOR_POLICY()` function to verify changed settings. For example, the following query retrieves a brief statement about the retention policy for the `NodeState` component:

```
=> SELECT GET_DATA_COLLECTOR_POLICY('NodeState');  
GET_DATA_COLLECTOR_POLICY  
-----  
50KB kept in memory, 250KB kept on disk.  
(1 row)
```

Tip: If you do not know the name of a component, you can query the [V_MONITOR.DATA_COLLECTOR](#) system table to get a full list. For example, the following query returns all current Data Collector components and a description of each: => `SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;`

See Also

- [GET_DATA_COLLECTOR_POLICY](#)
- [SET_DATA_COLLECTOR_POLICY](#)

Working with Data Collection Logs

Upon startup, a Vertica database creates a DataCollector directory within the /catalog directory.

The DataCollector directory holds the disk-based data collection logs, where retained data is kept in files named `<component>.<timestamp>.log`. Vertica might maintain several log files, per component, at any given time. See [Querying Data Collector Tables](#) for an example of how to view this information.

Also upon startup, Vertica creates two additional files, per component, in the DataCollector directory. These are SQL files that contain examples on how to load Data Collector data into another Vertica instance. These files are:

- `CREATE_<component>_TABLE.sql` — contains the SQL DDL needed to create a table into which Data Collector logs for the component can be loaded.
- `COPY_<component>_TABLE.sql` — contains example SQL to load (using [COPY](#) commands) the data log files into the table that the CREATE script creates.

Two functions let you manipulate these log files.

If you want to ...	See these topics in the SQL Reference Manual
Clear memory and disk records from Data Collector retention and reset collection statistics	CLEAR_DATA_COLLECTOR()

If you want to ...	See these topics in the SQL Reference Manual
Flush the Data Collector logs	FLUSH_DATA_COLLECTOR()
Retrieve a list of all current Data Collector components	V_MONITOR.DATA_COLLECTOR

Clearing the Data Collector

If you want to clear the Data Collector of all memory and disk records and reset the collection statistics in the V_MONITOR.DATA_COLLECTOR system table, call the CLEAR_DATA_COLLECTOR() function. You can clear records for all components on all nodes or you can specify individual components, one at a time.

To clear records on a single component, pass the function the component argument. For example, the following command clears records for the ResourceAcquisitions component only, returning a result of CLEAR (success):

```
=> SELECT clear_data_collector('ResourceAcquisitions');
clear_data_collector
-----
CLEAR
(1 row)
```

The following command clears the Data Collector records for all components on all nodes:

```
=> SELECT clear_data_collector();
clear_data_collector
-----
CLEAR
(1 row)
```

Note: After you clear the DataCollector log, the information is no longer available for querying.

Flushing Data Collector Logs

If you want to flush Data Collector information for all components or for an individual component, use the FLUSH_DATA_COLLECTOR() function. This function waits until memory logs are moved to disk and flushes the Data Collector, synchronizing the log with the disk storage:

To flush data collection for all components on all nodes:

```
=> SELECT flush_data_collector();
flush_data_collector
-----
FLUSH
(1 row)
```

To flush records on a single component, pass a component argument to the function. For example, the following command flushes the ResourceAcquisitions component:

```
=> SELECT flush_data_collector('ResourceAcquisitions');
flush_data_collector
-----
FLUSH
(1 row)
```

See Also

- [Data Collector Functions](#)
- [DATA_COLLECTOR](#)

Monitoring Data Collection Components

Query the V_MONITOR.DATA_COLLECTOR system table to get a list of Data Collector components, their current retention policies, and statistics about how much data is retained and how much has been discarded for various reasons. DATA_COLLECTOR also calculates the approximate collection rate, to aid in sizing calculations.

The following is a simple query that returns all the columns in this system table. See [V_MONITOR.DATA_COLLECTOR](#) in the SQL Reference Manual for additional details.

```
=> \xExpanded display is on.
=> SELECT * FROM data_collector;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
component          | AllocationPoolStatistics
table_name         | dc_allocation_pool_statistics
description        | Information about global memory pools, ...
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
disk_size_kb       | 256
record_too_big_errors | 0
lost_buffers       | 0
lost_records       | 0
retired_files      | 120
retired_records    | 53196
current_memory_records | 0
```

```
current_disk_records | 1454
current_memory_bytes | 0
current_disk_bytes   | 214468
first_time           | 2011-05-26 12:25:52.001109-04
last_time            | 2011-05-26 12:31:55.002762-04
kb_per_day           | 49640.4151810525
-[ RECORD 2 ]-----+-----
...
```

Related Topics

[V_MONITOR.DATA_COLLECTOR](#) and [Data Collector Functions](#) in the SQL Reference Manual
[Retaining Monitoring Information](#) and [How Vertica Calculates Database Size](#) in this guide

Querying Data Collector Tables

Caution: Data Collector tables (prefixed by `dc_`) reside in the `V_INTERNAL` schema and are provided for informational purposes only. They are provided as-is and are subject to removal or change without notice. If you use Data Collector tables in scripts or monitoring tools, you might need to change your scripts and tools after a Vertica upgrade. Micro Focus recommends that you use the [Workload Analyzer](#) instead of accessing the Data Collector tables directly.

Here's one useful example you can use to check on the state of your database. Upon startup, the Vertica database creates, under its catalog directory, a DataCollector directory. This directory holds the disk-based data collection logs. The main data is kept in files named `<component>.<timestamp>.log`.

When you start your database an entry is created in the `dc_startups` table. The following is the result of querying this table.

```
=> SELECT * FROM dc_startups;
-[ RECORD 1 ]-----+-----
time           | 2011-05-26 17:35:40.588589-04
node_name      | v_vmartdb_node0001
version        | Vertica Analytic Database v5.0.4-20110526
command_line   | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/
               | catalog/vmartdb/v_vmartdb_node0001_catalog -h
               | 10.10.50.123 -p 5608
codename       | 5.0
build_tag      | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu
               | May 26 3:37:18 2011' $BuildId$
build_type     | 64-bit Optimized Build
compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale  | UTF-8
```

```
database_path | /home/vmartdb/catalog/vmartdb/v_vmartdb_node0001_catalog
alt_host_name | 10.10.50.123
alt_node_name |
start_epoch  |
-[ RECORD 2 ]-----+-----
time         | 2011-05-26 17:35:40.218999-04
node_name    | v_vmartdb_node0004
version      | Vertica Analytic Database v5.0.4-20110526
command_line | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/
              | catalog/vmartdb/v_vmartdb_node0004_catalog -h
              | 10.10.50.126 -p 5608
codename     | 5.0
build_tag    | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu
              | May 26 3:37:18 2011' $BuildId$
build_type   | 64-bit Optimized Build
compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale | UTF-8
database_path | /home/vmartdb/catalog/vmartdb/v_vmartdb_node0004_catalog
alt_host_name | 10.10.50.126
alt_node_name |
start_epoch  |
-[ RECORD 3 ]-----+-----
time         | 2011-05-26 17:35:40.931353-04
node_name    | v_vmartdb_node0003
version      | Vertica Analytic Database v5.0.4-20110526
command_line | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/
              | catalog/vmartdb /v_vmartdb_node0003_catalog -h
              | 10.10.50.125 -p 5608
codename     | 5.0
build_tag    | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu
              | May 26 3:37:18 2011' $BuildId$
build_type   | 64-bit Optimized Build
compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale | UTF-8
database_path | /home/vmartdb/catalog/vmartdb/v_vmartdb_node0003_catalog
alt_host_name | 10.10.50.125
alt_node_name |
start_epoch  |
-[ RECORD 4 ]-----+-----
```

Monitoring Query Plan Profiles

See [Profiling Query Plans](#).

Monitoring Partition Reorganization

When you use `ALTER TABLE . . . REORGANIZE`, the operation reorganizes the data in the background.

You can monitor details of the reorganization process by polling the following system tables:

- [V_MONITOR.PARTITION_STATUS](#) displays the fraction of each table that is partitioned correctly.
- [V_MONITOR.PARTITION_REORGANIZE_ERRORS](#) logs errors issued by the reorganize process.
- [V_MONITOR.PARTITIONS](#) displays NULL in the `partition_key` column for any ROS that was not reorganized.

Note: The corresponding foreground process to `ALTER TABLE . . . REORGANIZE` is [PARTITION_TABLE](#).

Using Queries to Monitor Resource Pool Size and Usage

You can use the [Linux top command](#) to determine the overall CPU usage and I/O wait time across the system. However, because of file system caching, the resident memory size indicated by `top` is not a good indicator of actual memory use or available reserve.

Instead, Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests, and the state of the queues.

The [RESOURCE_POOLS](#) table lets you view various resource pools defined in the system (both internal and user-defined), and the [RESOURCE_POOL_STATUS](#) table lets you view the current state of the resource pools.

Examples

This example shows how to find the various resource pools defined in the system.

```
=> SELECT name, memorysize, maxmemorysize FROM V_CATALOG.RESOURCE_POOLS;
```

name	memorysize	maxmemorysize
general		Special: 95%
sysquery	64M	
sysdata	100M	10%
wosdata	0%	25%
tm	200M	
refresh	0%	

```

recovery | 0%      |
dbd      | 0%      |
jvm      | 0%      | 10%
(9 rows)

```

Viewing Only User-Defined Resource Pools

To see only the user-defined resource pools, you can limit your query to return records where `IS_INTERNAL` is false.

Note: The user-defined pools shown in these examples also appear in subsequent sections related to Workload Management.

This example shows how to find information on user-defined resource pools:

```

=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency
      FROM V_CATALOG.RESOURCE_POOLS where is_internal = 'f';

```

name	memorysize	maxmemorysize	priority	maxconcurrency
load_pool	0%		10	
ceo_pool	250M		10	
ad hoc_pool	200M	200M	0	
billing_pool	0%		0	3
web_pool	25M		10	5
batch_pool	150M	150M	0	10
dept1_pool	0%		5	
dept2_pool	0%		8	

(8 rows)

Viewing the Status of All Resource Pools

The following example shows how to access the `V_MONITOR.RESOURCE_POOL_STATUS` table to return the current state of all resource pools on node0001:

```

=>\x
Expanded display is on
=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb,
      general_memory_borrowed_kb, running_query_count
      FROM V_MONITOR.RESOURCE_POOL_STATUS where node_name ilike '%node0001';

```

pool_name	general
[RECORD 1]	

```

memory_size_kb          | 2983177
memory_size_actual_kb  | 2983177
memory_inuse_kb        | 0
general_memory_borrowed_kb | 0
running_query_count    | 0
-[ RECORD 2 ]-----+-----
pool_name              | sysquery
memory_size_kb        | 65536
memory_size_actual_kb | 65536
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 3 ]-----+-----
pool_name              | sysdata
memory_size_kb        | 102400
memory_size_actual_kb | 102400
memory_inuse_kb       | 4096
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 4 ]-----+-----
pool_name              | wosdata
memory_size_kb        | 0
memory_size_actual_kb | 0
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 5 ]-----+-----
pool_name              | tm
memory_size_kb        | 204800
memory_size_actual_kb | 204800
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 6 ]-----+-----
pool_name              | refresh
memory_size_kb        | 0
memory_size_actual_kb | 0
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 7 ]-----+-----
pool_name              | recovery
memory_size_kb        | 0
memory_size_actual_kb | 0
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 8 ]-----+-----
pool_name              | dbd
memory_size_kb        | 0
memory_size_actual_kb | 0
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0
running_query_count   | 0
-[ RECORD 9 ]-----+-----
pool_name              | jvm
memory_size_kb        | 0
memory_size_actual_kb | 0
memory_inuse_kb       | 0
general_memory_borrowed_kb | 0

```

```
running_query_count      | 0
```

Viewing Query Resource Acquisitions

The following example displays all resources granted to the queries that are currently running. The information shown is stored in the `V_MONITOR.RESOURCE_ACQUISITIONS` table. You can see that the query execution used 708504 KB of memory from the GENERAL pool.

```
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
       queue_entry_timestamp, acquisition_timestamp
       FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

-[ RECORD 1 ]-----+-----
pool_name      | sysquery
thread_count   | 4
open_file_handle_count | 0
memory_inuse_kb | 4103
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05
acquisition_timestamp | 2013-12-05 07:07:08.815367-05
-[ RECORD 2 ]-----+-----
...
-[ RECORD 8 ]-----+-----
pool_name      | general
thread_count   | 12
open_file_handle_count | 18
memory_inuse_kb | 708504
queue_entry_timestamp | 2013-12-04 12:55:38.566614-05
acquisition_timestamp | 2013-12-04 12:55:38.566623-05
-[ RECORD 9 ]-----+-----
...
```

You can determine how long a query waits in the queue before it can run. To do so, you obtain the difference between the `acquisition_timestamp` and the `queue_entry_timestamp` using a query as this example shows:

```
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,
       (acquisition_timestamp-queue_entry_timestamp) AS 'queue wait'
       FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

-[ RECORD 1 ]-----+-----
pool_name      | sysquery
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05
acquisition_timestamp | 2013-12-05 07:07:08.815367-05
queue wait     | 00:00:00.000005
-[ RECORD 2 ]-----+-----
pool_name      | sysquery
queue_entry_timestamp | 2013-12-05 07:07:14.714412-05
acquisition_timestamp | 2013-12-05 07:07:14.714417-05
queue wait     | 00:00:00.000005
-[ RECORD 3 ]-----+-----
pool_name      | sysquery
queue_entry_timestamp | 2013-12-05 07:09:57.238521-05
```

```
acquisition_timestamp | 2013-12-05 07:09:57.281708-05  
queue wait           | 00:00:00.043187  
-[ RECORD 4 ]-----+-----  
...
```

See Also

- See the [SQL Reference Manual](#) for detailed descriptions of the monitoring tables.
- See [Monitoring Resource Pools](#) for descriptions of other ways to monitor resource usage.

Monitoring Recovery

There are several ways to monitor database recovery:

- [Log files on each host](#)
- [Admintools \(View Database Cluster State\)](#)
- [System tables](#)
- [Cluster Status after recovery](#)

Viewing Log Files on Each Node

During database recovery, Vertica adds logging information to the `vertica.log` on each host. Each message is identified with a `[Recover]`string.

Use the `tail` command to monitor recovery progress by viewing the relevant status messages, as follows.

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log  
01/23/08 10:35:31 thr:Recover:0x2a98700970 [Recover] <INFO> Changing host v_vmart_node0001 startup  
state from INITIALIZING to RECOVERING  
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Recovering to specified epoch 0x120b6  
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running 1 split queries  
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running query: ALTER PROJECTION proj_  
tradesquotes_0 SPLIT v_vmart_node0001 FROM 73911;
```

Using System Tables to Monitor Recovery

Use the following system tables to monitor recover:

- [RECOVERY_STATUS](#)
- [PROJECTION_RECOVERIES](#)

Specifically, the `recovery_status` system table includes information about the node that is recovering, the epoch being recovered, the current recovery phase, and running status:

```
=>select node_name, recover_epoch, recovery_phase, current_completed, is_running from recovery_status;
```

node_name	recover_epoch	recovery_phase	current_completed	is_running
v_vmart_node0001			0	f
v_vmart_node0002	0	historical pass 1	0	t
v_vmart_node0003	1	current	0	f

The `projection_recoveries` system table maintains history of projection recoveries. To check the recovery status, you can summarize the data for the recovering node, and run the same query several times to see if the counts change. Differing counts indicate that the recovery is working and in the process of recovering all missing data.

```
=> select node_name, status , progress from projection_recoveries;
```

node_name	status	progress
v_vmart_node0001	running	61

To see a single record from the `projection_recoveries` system table, add `limit 1` to the query.

After a recovery has completed, Vertica continues to store information from the most recent recovery in these tables.

Viewing Cluster State and Recovery Status

Use the admintools `view_cluster` tool from the command line to see the cluster state:

```
$ /opt/vertica/bin/admintools -t view_cluster
```

DB	Host	State
<data_base>	112.17.31.10	RECOVERING
<data_base>	112.17.31.11	UP
<data_base>	112.17.31.12	UP

```
<data_base> | 112.17.31.17 | UP
```

Monitoring Cluster Status After Recovery

When recovery has completed:

1. Launch Administration Tools.
2. From the Main Menu, select **View Database Cluster State** and click **OK**.

The utility reports your node's status as UP.

Note: You can also monitor the state of your database nodes on the Management Console Overview page under the Database section, which tells you the number of nodes that are up, critical, recovering, or down. To get node-specific information, click Manage at the bottom of the page.

Clearing PROJECTION_REFRESHES History

Information about a refresh operation—whether successful or unsuccessful—is maintained in system table `PROJECTION_REFRESHES` until the function `CLEAR_PROJECTION_REFRESHES` executes or the storage quota for the table is exceeded. The table's `IS_EXECUTING` column returns a boolean value that indicates whether the refresh is running now (t) or occurred earlier (f).

To immediately purge this information, use the `CLEAR_PROJECTION_REFRESHES()` function:

```
=> SELECT clear_projection_refreshes();
clear_projection_refreshes
-----
CLEAR
(1 row)
```

Note: Only the rows where the `PROJECTION_REFRESHES.IS_EXECUTING` column equals *false* are cleared.

See Also

- [CLEAR_PROJECTION_REFRESHES](#)
- [PROJECTION_REFRESHES](#)

Monitoring Vertica Using Notifiers

A Vertica notifier is a push-based mechanism capable of sending messages from Vertica to endpoints such as Apache Kafka. For example, you could configure a long-running script to send notifications at various stages and then at the completion of a task.

To use a notifier, follow these steps:

1. Create a notifier with [CREATE NOTIFIER](#).
2. Use the notifier to send messages from Vertica to the end point with the meta-function [NOTIFY](#).

See Also

[ALTER NOTIFIER](#)

Backing Up and Restoring the Database

Creating regular database backups is an important part of basic maintenance tasks at your site. Vertica supplies a comprehensive utility, called `vbr`, for this purpose. This utility lets you back up, restore, list backups, and copy your database to another cluster. You can create full and incremental database backups, and backups of schemas or tables (object-level backups) for use with a multi-tenanted database. When a full backup exists, you can restore one or more objects from the backup or the entire database, as required.

Using `vbr`, you can save your data to a variety of locations:

- A local directory on the nodes in the cluster
- One or more hosts outside of the cluster
- A different Vertica cluster (effectively cloning your database)
- Amazon S3 storage

Impact of Backups on Vertica Nodes

While a backup is taking place, the backup process can consume additional storage. The amount of space consumed depends on the size of your catalog and any objects that you drop during the backup. The backup process releases this storage once the backup is complete.

Compatibility Requirements for Restore and Replication

Creating backups with `vbr` requires restoring backups with the same utility. Vertica 6.x and later supports object-level backups.

Vertica supports restore, replication, and `copycluster` actions only to the same exact version of Vertica that created the backup. For example, you cannot restore, replicate objects, or `copycluster` a version 7.0.2-3 backup to a version 7.0.2-5 database.

Additional Considerations for HDFS Storage Locations

If your database has any storage locations on HDFS, you must do additional configuration to enable those storage locations to be backed up. See [Backing Up HDFS Storage Locations](#) in [Integrating with Apache Hadoop](#).

Supported File Systems

Vertica supports the following file systems for backup and temporary directory locations:

- ext3
- ext4
- NFS
- Amazon S3 Standard

The file system at your backup and temporary directory locations must support `fcntl` lockf (POSIX) file locking. [Backups to Amazon S3](#) use a secondary location to handle file locking.

Creating vbr Configuration Files

The vbr utility uses a configuration file for the information required to back up and restore a full- or object-level backup or copy a cluster. You cannot run vbr without a configuration file because no default file exists. You can, however:

- Create a configuration file. See [Start the vbr Configuration Script](#).
- Copy and edit an existing configuration file and save the changes to a different file name.
- Copy and edit one of the [sample configuration files](#) included with Vertica.

Note: You must be logged on as the database administrator, not root, to create the vbr configuration file.

Start the vbr Configuration Script

To create a configuration file, enter this command:

```
> /opt/vertica/bin/vbr --setupconfig
```

The script prompts you to answer several questions, as shown in the following summarized example:

```
[dbadmin@localhost ~]$ /opt/vertica/bin/vbr --setupconfig
Snapshot name (backup_snapshot): fullbak1
Number of restore points (1): 3
Specify objects (no default):
Object restore mode (coexist, createOrReplace or create) (createOrReplace):
Vertica user name (dbadmin):
Save password to avoid runtime prompt? (n) [y/n]: y
Password to save in vbr config file (no default):
Node v_vmart_node0001
Backup host name (no default): 194.66.82.11
Backup directory (no default): /home/dbadmin/backups
Config file name (fullbak1.ini):
Password file name (no default value) (no default):pwdfile
Change advanced settings? (n) [y/n]: n
Saved vbr configuration to fullbak1.ini.
Saved vbr database password to pwdfile.ini.
```

For further information on how to answer these questions, see:

See the following instructions to complete the questions in the configuration script:

- Specify a Backup Name
- Specify Restore Points
- Specify Object Restore Mode
- Specify Full or Object-Level Backups
- Enter the User Name
- Save the Account Password
- Specify the Backup Host and Directory
- Save the Configuration File
- Continue to Advanced Settings

Specify a Backup Name

Specify the name of the backup for `vbr` to create:

```
Snapshot name (backup_snapshot):
```

You must provide a name for the backup that clearly identifies the specific backup. The backup name is used in the directory tree structure that `vbr` creates for each node. Because Vertica automatically augments the backup subdirectories with date and time indicators, do not add dates to the backup name.

Create different configuration files with specific backup names for full and object-level backups, but use the same backup directory for both types of backups. For example, the configuration file for a full database backup, named `fullbak.ini`, has these `snapshotName` and `backupDir` parameter values:

```
snapshotName=fullbak  
backupDir=/home/dbadmin/data/backups
```

The configuration file for the object-level backup, named `objectbak.ini`, has these parameter values:

```
snapshotName=objectbak  
backupDir=/home/dbadmin/data/backups
```

Specify Restore Points

Specify how many backups to save as a number of restore points:

```
Number of restore points (1):
```

The default value is 1, indicating that `vbr` always retains one additional restore point.

Saving multiple restore points gives you the option to recover from one of several backups. For example, if you specify 3, you have 1 current backup, and 3 backup archives. Vertica stores the value you enter as the `restorePointLimit` parameter in the `vbr` configuration file.

Specify Object Restore Mode

Specify how Vertica should handle restored objects, as this example shows:

```
Object restore mode (coexist, createOrReplace or create)
(createOrReplace):
```

Vertica supports the following object restore modes:

- `createOrReplace` (default) — Vertica creates any objects that do not exist. If the object does exist, `vbr` overwrites it with the version from the archive.
- `create` — Vertica creates any objects that do not exist. If an object being restored does exist, Vertica displays an error message and skips that object.
- `coexist` — Vertica creates all restored objects with the form `<backup>_<timestamp>_<object_name>`. This approach allows existing and restored objects to exist simultaneously. If the appended information pushes the schema name past the maximum length of 128 characters, HP Vertica truncates the name. You can perform a reverse lookup of the original schema name by querying the system table [TRUNCATED_SCHEMATA](#).

In all modes, Vertica restores data with the current epoch. Object restore mode settings do not apply to backups and full restores.

Specify Full or Object-Level Backups

Indicate whether you want to create a full- or object-level backup:

Specify objects (no default):

- For full database backup: Press Enter to continue creating a configuration file for a full database backup.
- For object-level backup: Enter the names of any schema or table for which you want to create an object-level backup configuration file. Enter table names in the form `schema.objectname`.

For example, to make backups of the table, `customers` from the schema, `finance`, enter `finance.customers`. Separate each name with a comma (,). The objects you enter appear listed in the `Objects` parameter of the configuration file.

Enter the User Name

Enter the user name of the person who invoking `vbr`:

Vertica user name (dbadmin):

Note: This user name must be the user identified as the database administrator for the entire database. No other user has permission to invoke this action, even if the user has `dbadmin` permissions within portions of the database.

Save the Account Password

Specify whether `vbr` prompts for an account password at run time:

Save password to avoid runtime prompt? (n) [y/n]:

Press **Enter** to accept the default (n) and continue. Because you are not saving the password, you must enter it when you run `vbr`.

To save your password to a [password configuration file](#), enter `y`. The utility prompts for you to enter the password but does not display the text as you type:

Save password to avoid runtime prompt? (n) [y/n]:

The parameter name in the configuration file that indicates whether to prompt for a password is `dbPromptForPassword`.

Specify the Backup Host and Directory

Vbr lists each node name for your cluster. Enter the backup host name and directory for each cluster node at the prompts:

```
Node v_vmart_node0001  
Backup host name (no default):  
Backup directory (no default):
```

No defaults exist for either the host name or the backup directory.

In the configuration file, all cluster nodes appear listed by name under the [Mapping] section. Each cluster node contains a line that includes the cluster node, backup host, and backup directory location:

```
[Mapping]  
v_vmart_node0001 = 194.66.82.11:/home/dbadmin/backups
```

Save the Configuration File

To save the configuration file with its default name, enter a configuration file name of your choice, or press (Enter):

```
Config file name (fullbak.ini):
```

The default name consists of:

- The backup name you supplied (fullbak in this example)
- An .ini file extension

The vbr utility confirms that it saved the file:

```
Saved vbr configuration to fullbak.ini.
```

By default, Vertica saves the file in the current working directory. You can include a fully qualified or relative path to save the configuration file to an alternate location.

Note: The backup configuration file is typically stored on the cluster you are backing up. Therefore, Micro Focus recommends that you also save a copy of the configuration file on the backup host. If you do so, the configuration file remains available, even if the cluster

```
node is lost.
```

Continue to Advanced Settings

To continue with advanced configuration settings, enter `y`:

```
Change advanced settings? (n) [y/n]:
```

To continue *without* advanced settings and save the configuration file with the information you just entered, press **Enter**.

Changing the `objectRestoreMode` Parameter Value

After you create the configuration file, you can also specify an `objectRestoreMode` parameter. The `objectRestoreMode` parameter is associated only with restoring object-level backups.

Related Tasks

- [Configuring Advanced VBR Options](#)

Configuring Required VBR Parameters

To invoke the `vbr` utility to create a configuration file, enter this command:

```
> /opt/vertica/bin/vbr --setupconfig
```

The utility prompts you to answer the following questions. Each is followed by its default value, if any exists. Type **Enter** to accept a default value.

The second table column lists the name of the parameter associated with the question, as it appears in the configuration file.

Setupconfig Prompt	Parameter Name
Snapshot name: (backup_snapshot)	snapshotName
Number of restore points? (1):	restorePointLimit

Specify objects (no default):	objects
Object restore mode (coexist, createOrReplace or create)(createOrReplace):	objectRestoreMode
Vertica user name (current user):	dbUser
Save password to avoid runtime prompt (n) [y/n]	dbPromptForPassword
Database user password to save in vbr password config file (no default):	dbPassword
Backup host name (no default):	Backup host
Backup directory (no default):	Backup directory
Config file name (backup_snapshot.ini):	N/A
Password file name (no default):	PasswordFile
Change advanced settings? (n) [y/n]:	N/A

To change any advanced parameters, respond to the last question, Change advanced settings? by entering y.

After you successfully complete all of the required questions, vbr generates a configuration file with the information you supplied. Create separate configuration files for a full backup and each object-level backup. Use distinct backup names in each configuration file.

When the setup completes processing, enter a configuration file name. Use this file name when you run the `--task backup` or other commands. The utility uses the configuration file contents for both backup and restore tasks, and also for the `--copycluster` task.

If you do not successfully complete all of the required questions, vbr lists error messages and hints but does not create a configuration file. You can then run the `--setupconfig` command again to specify any missing or incorrect information.

Sample Session Configuring Required Parameters

The following sample test session shows the required configuration file parameters needed to create a configuration file. The utility detects the Vertica node names in the cluster, so you do not have to supply them (v_example_node0001, v_example_node0002, and v_exampleledb_node0003):

```
> /opt/vertica/bin/vbr --setupconfig
Snapshot name (snapshotName): ExampleBackup
Backup Vertica configurations? (n) [y/n] y
Number of restore points? (1): 5
Specify objects (no default): dim, dim2
Object restore mode (coexist, createOrReplace or create) (createOrReplace): coexist
```

```
Vertica user name (current_user): dbadmin
Save password to avoid runtime prompt? (n)[y/n]: y
Database user password to save in vbr password config file (no default): mypw234
Node v_example_node0001
Backup host name (no default): backup01
Backup directory (no default): /home/dbadmin/backups
Node v_example_node0002
Backup host name: backup02
Backup directory: /home/dbadmin/backups
Node v_example_node0003
Backup host name: backup03
Backup directory: /home/dbadmin/backups
Config file name: exampleBackup.ini
Password file name: StoredPasswords
Change advanced settings? (n)[y/n]: n
Saved vbr configuration to ExampleBackup.ini.
```

See Also

- [VBR Configuration File Reference](#)

Related Tasks

- [Creating vbr Configuration Files](#)
- [Copying the Database to Another Cluster](#)

Configuring Advanced VBR Options

When you use `vbr` to create your configuration file, you can configure advanced parameters after configuring the basic ones that are required. The first time you invoke this utility, enter the command:

```
> /opt/vertica/bin/vbr --setupconfig
```

After you complete the required parameter options, continue to advanced settings. Respond to the last question by entering `y` in response:

```
Change advanced settings? (n)[y/n]: y
```

The completed configuration file includes the following list of the advanced parameters, their description, and their default values:

Configuration Parameter	Description and Default Value
bwlimit	(Deprecated) The transfer bandwidth limit in kilobytes per second for a backup or a restore operation. This parameter is replaced by total_bwlimit_backup and total_bwlimit_restore.
checksum	When set to y, this parameter uses checksum for data integrity instead of file date and size. The default value is n.
concurrency_backup	The maximum number of backup TCP rsync connections per node. The default value is 1.
concurrency_restore	The maximum number of restore TCP rsync connections per node. The default value is 1.
encrypt	When set to y, encrypts data during transmission. The default value is n.
port_rsync	The Port number for the rsync daemon. The default value is 50000.
retryCount	The number of times to retry if a connection attempt fails. The default value is 2.
retryDelay	The number of seconds to wait between connection retry attempts. The default value is 1.
serviceAccessUser	The user name used for simple authentication of rsync connections. Vertica stores the password for this user name in a separate configuration file .
tempDir	The temporary directory (/tmp/vbr). For additional information about the tempDir configuration parameter, see [Misc] Miscellaneous Settings .
total_bwlimit_backup	The total bandwidth limit in kilobytes per second per node for backup connections. Vertica distributes this bandwidth among the number of connections set in concurrency_backup. The total network load allowed by this value is the number of nodes multiplied by the value of this parameter. For example, a three node cluster and a total_bwlimit_backup value of 100 would allow 300Kbytes/sec of network traffic. The default value of 0 allows unlimited bandwidth.

Configuration Parameter	Description and Default Value
total_bwlimit_restore	The total bandwidth limit in kilobytes per second per node for restore connections. Vertica distributes this bandwidth among the number of connections set in <code>concurrency_restore</code> . The total network load allowed by this value is the number of nodes multiplied by the value of this parameter. For example, a three node cluster and a <code>total_bwlimit_restore</code> value of 100 would allow 300Kbytes/sec of network traffic. The default value of 0 allows unlimited bandwidth.

Example of Configuring Advanced Parameters

This example shows how you might set some of the advanced configuration file parameters to configure your backup and restore settings:

```
Change advanced settings? (n)[y/n]: y
Temp directory (/tmp/vbr):
Number of times to retry backup? (2): 5
Seconds between retry attempts? (1): 3
Encrypt data during transmission? (n) [y/n] n
Use checksum for data integrity (not file date and size)? (n)[y/n]: n
Port number for Rsync daemon (50000):
User name to access rsync daemon (no default): rsyncaccount
Password of the user who accesses rsync daemon:
Transfer bandwidth limit in KBPS or 0 for unlimited (0): 0
Saved vbr configuration to exampleBackup.ini.
```

See Also

- [VBR Configuration File Reference](#)

Related Tasks

- [Configuring Backup Hosts](#)

Configuring the Hard-Link Local Parameter

Creating hard link local backups requires that you manually add the `hardLinkLocal=True` parameter to the `[Transmission]` section of the `vbr` configuration file.

Any configuration file you generate with the `vbr --setupconfig` always includes a `[Transmission]` section. If you have an existing `vbr` configuration file (*backup_name.ini*), add the `hardLinkLocal` parameter to the `[Transmission]` section.

Generating a Configuration File

To generate a configuration file *without* advanced options:

1. Add the parameter as the sole entry in the `[Transmission]` section:

```
[Transmission]
hardLinkLocal = True
```

2. Save the configuration file.

You can also generate a configuration file *with* advanced options.

1. Add the `hardLinkLocal` parameter as the last entry in the `[Transmission]` section:

```
[Transmission]
encrypt = False
checksum = False
port_rsync = 50000
total_bwlimit_backup = 0
total_bwlimit_restore = 0
hardLinkLocal = True
```

2. Save the configuration file.

Restrictions for the Backup Encryption Option

You cannot use `encrypt` parameter advanced option when creating a hard link local backup. If you add `hardLinkLocal=true` to a configuration file that includes `encrypt=true`, `vbr` issues a warning and then ignores the encryption parameter.

Configuring a Local Backup File Without Hard Links

To create a local backup without hard file links, omit the `hardLinkLocal=True` parameter from the configuration file. Specify the `backupDir` parameter as a location on the same file system as the database catalog and data files. Then, the `vbr` utility creates a backup by copying the files, even when they are located on the same file system.

Example Password Configuration File

The following example password configuration file shows the options for configuring VBR passwords. If you have chosen not to store passwords, this file remains empty. Only the `dbadmin` or members of that user's permission group can view the contents of the file.

```
[Passwords]
dbPassword = DBsecurity
serviceAccessPass = rsyncpwd
;Specifies password for remote database. Used only for restoring to alternate cluster.
dest_dbPassword = DestinationPwd
```

Sample VBR .ini Files

To assist users with the task of configuring the `vbr` utility, Vertica includes sample configuration files that you can copy, edit, and deploy for your various `vbr` tasks. Vertica automatically installs these files at `/opt/vertica/share/vbr/example_configs`.

Vertica includes the following files:

- [Database Copy to an Alternate Cluster](#) - `copycluster.ini`
- [External Central Backup/Restore](#) - `backup_restore_full_external.ini`
- [Full Hardlink Backup/Restore](#) - `backup_restore_full_hardlink.ini`
- [Full Local Backup/Restore](#) - `backup_restore_full_local.ini`
- [Object-Level Local Backup/Restore](#) - `backup_restore_object_local.ini`
- [Object Replication to an Alternate Database](#) - `replicate.ini`

- [Full Backup and Restore to an Alternate Cluster](#) - restore_to_other_cluster.ini
- [Backup/Restore To Amazon S3](#) - backup_restore_s3.htm

Database Copy to an Alternate Cluster

```
; This sample vbr configuration file is configured for the copycluster vbr task.
; Copycluster supports full database copies only, not specific objects.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[Mapping]
; For each node of the source database, there must be a [Mapping] entry specifying the corresponding
hostname of the destination database node.
; !!Mandatory!! node_name = new_host/ip (no defaults)
v_exampledb_node0001 = new_host1.example
v_exampledb_node0002 = new_host2.example
v_exampledb_node0003 = new_host3.example
v_exampledb_node0004 = new_host4.example
; v_exampledb_node0001 = 10.0.90.17
; v_exampledb_node0002 = 10.0.90.18
; v_exampledb_node0003 = 10.0.90.19
; v_exampledb_node0004 = 10.0.90.20

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to copy.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between retry attempts, if a failure occurs.
```

```
; retryDelay = 1

; Full path to the password configuration file containing database password credentials
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; The maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all copycluster connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bwlimit_backup = 0

; The maximum number of copycluster TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bwlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator
; dbUser = current_username
```

External Central Backup/Restore

```
; This sample vbr configuration file shows full or object backup and restore to a separate remote
backup-host for each respective database host.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[Mapping]
; !!Mandatory!! This section defines what host and directory will store the backup for each node.
; node_name = backup_host:backup_dir
```

```
; In this "parallel backup" configuration, each node backs up to a distinct external host.
; To backup all database nodes to a single external host, use that single hostname/IP address in each
entry below.
v_exampledb_node0001 = 10.20.100.156:/home/dbadmin/backups
v_exampledb_node0002 = 10.20.100.157:/home/dbadmin/backups
v_exampledb_node0003 = 10.20.100.158:/home/dbadmin/backups
v_exampledb_node0004 = 10.20.100.159:/home/dbadmin/backups
#v_exampledb_node0001 = example1.net1:/home/dbadmin/backups
#v_exampledb_node0002 = example1.net2:/home/dbadmin/backups
#v_exampledb_node0003 = example1.net3:/home/dbadmin/backups
#v_exampledb_node0004 = example1.net4:/home/dbadmin/backups

[Misc]
; !!Recommended!! Snapshot name. Object and full backups should always have different snapshot
names.
; Backups with the same snapshotName form a time sequence limited by restorePointLimit.
; Valid characters: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if some error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When enabled, Vertica confirms that the specified backup locations contain
; sufficient free space and inodes to allow a successful backup. If a backup
; location has insufficient resources, Vertica displays an error message and
; cancels the backup. If Vertica cannot determine the amount of available space
; or number of inodes in the backupDir, it displays a warning and continues
; with the backup.
```

```
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all backup connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bwlimit_backup = 0

; The maximum number of backup TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bwlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator
; dbUser = current_username
```

Full Hardlink Backup/Restore

```
; This sample vbr configuration file shows backup and restore using hard-links to data files on each
; database host for that host's backup.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[Mapping]
; For each database node there must be one [Mapping] entry to indicate the directory to store the
; backup.
; !!Mandatory!! Backup host name (no default) and Backup directory (no default).
; node_name = backup_host:backup_dir
; Must use [] for hardlink backups
```

```
v_exampledb_node0001 = [ ]:/home/dbadmin/backups
v_exampledb_node0002 = [ ]:/home/dbadmin/backups
v_exampledb_node0003 = [ ]:/home/dbadmin/backups
v_exampledb_node0004 = [ ]:/home/dbadmin/backups

[Misc]
; !!Recommended!! Snapshot name. Object and full backups should always have different snapshot
names.
; Backups with the same snapshotName form a time sequence limited by restorePointLimit.
; Valid characters: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True

[Transmission]
; !!Mandatory!! Identifies the backup as a hardlink style backup.
hardLinkLocal = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
; dbName = current_database

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile =

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Specifies the number of backup attempts after an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts if a failure occurs.
; retryDelay = 1

; When enabled, Vertica confirms that the specified backup locations contain
; sufficient free space and inodes to allow a successful backup. If a backup
; location has insufficient resources, Vertica displays an error message and
; cancels the backup. If Vertica cannot determine the amount of available space
; or number of inodes in the backupDir, it displays a warning and continues
; with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum
```

```
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator.
; dbUser = current_username
```

Full Local Backup/Restore

```
; This is a sample vbr configuration file for backup and restore using a file system on each database
host for that host's backup.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[Mapping]
; !!Mandatory!! For each database node there must be one [Mapping] entry to indicate the directory to
store the backup.
; node_name = backup_host:backup_dir
; use [] for localhost
v_exampledb_node0001 = []:/home/dbadmin/backups
v_exampledb_node0002 = []:/home/dbadmin/backups
v_exampledb_node0003 = []:/home/dbadmin/backups
v_exampledb_node0004 = []:/home/dbadmin/backups

[Misc]
; !!Recommended!! Snapshot name
; Valid values: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;
```

```
[Misc]

; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if some error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When enabled, Vertica confirms that the specified backup locations contain
; sufficient free space and inodes to allow a successful backup. If a backup
; location has insufficient resources, Vertica displays an error message and
; cancels the backup. If Vertica cannot determine the amount of available space
; or number of inodes in the backupDir, it displays a warning and continues
; with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator
; dbUser = current_username
```

Full Backup and Restore to an Alternate Cluster

```
; This sample vbr configuration file shows full backup and restore to another cluster.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;
```

```
[Mapping]
; There must be one [Mapping] section for all of the nodes in your database cluster.
; !!Mandatory!! Backup host name (no default) and Backup directory (no default)
; node_name = backup_host:backup_dir
v_exampledb_node0001 = new_host0001:/home/dbadmin/backups
v_exampledb_node0002 = new_host0002:/home/dbadmin/backups
v_exampledb_node0003 = new_host0003:/home/dbadmin/backups
v_exampledb_node0004 = new_host0004:/home/dbadmin/backups

[NodeMapping]
; !!Recommended!! This section is required if node names are different between source and destination
databases.
v_exampledb_node0001 = new_host0001
v_exampledb_node0002 = new_host0002
v_exampledb_node0003 = new_host0003
v_exampledb_node0004 = new_host0004

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr prompts the user for database password every time.
; If False, specify location of password config file in 'passwordFile' parameter in [Misc] section.
; dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if some error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Full path to the password configuration file.
; Store this file in a directory only readable by the dbadmin.
; (no default)
; passwordFile =

; When enabled, Vertica confirms that the specified backup locations contain
; sufficient free space and inodes to allow a successful backup. If a backup
; location has insufficient resources, Vertica displays an error message and
; cancels the backup. If Vertica cannot determine the amount of available space
; or number of inodes in the backupDir, it displays a warning and continues
; with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600
```

```
[Transmission]
; Sets options for transmitting the data when using backup hosts.
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all backup connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bwlimit_backup = 0

; The maximum number of backup TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bwlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator.
; dbUser = current_username
```

Object-Level Local Backup/Restore

```
; This sample vbr configuration file shows object-level backup and restore
; using a file system on each database host for that host's backup.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; Option and values are separated by an equal sign.
; Only arguments marked as '!!Mandatory!!' must be specified explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[Mapping]
; There must be one [Mapping] section for all of the nodes in your database cluster.
; !!Mandatory!! Backup host name (no default) and Backup directory (no default)
; node_name = backup_host:backup_dir
; [] indicates backup to localhost
v_exampledb_node0001 = []:/home/dbadmin/backups
v_exampledb_node0002 = []:/home/dbadmin/backups
v_exampledb_node0003 = []:/home/dbadmin/backups
v_exampledb_node0004 = []:/home/dbadmin/backups

[Misc]
; !!Recommended!! Snapshot name. Object and full backups should always have different snapshot
names.
; Backups with the same snapshotName form a time sequence limited by restorePointLimit.
```

```
; Valid values: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

; Specifies which tables and/or schemas to backup.
; !!Mandatory!! objects for an object backup
; Specifies which tables and/or schemas to copy. For tables, the containing schema defaults to
public.
; (no default)
objects = mytable, myschema, myothertable

; Specifies how Vertica handles objects of the same name when restoring schema or table backups.
; objectRestoreMode = createOrReplace

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr will prompt user for database password every time.
; If changed to False, specify location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When enabled, Vertica confirms that the specified backup locations contain
; sufficient free space and inodes to allow a successful backup. If a backup
; location has insufficient resources, Vertica displays an error message and
; cancels the backup. If Vertica cannot determine the amount of available space
; or number of inodes in the backupDir, it displays a warning and continues
; with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
```

```
; SnapshotEpochLagFailureThreshold = 3600

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator.
; dbUser = current_username
```

Object Replication to an Alternate Database

```
; This sample vbr configuration file shows the replicate vbr task.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[Mapping]
; There must be one [Mapping] section for all of the nodes in your database cluster.
; !!Mandatory!! Target host name (no default)
; node_name = new_host
v_exampledb_node0001 = new_host0001
v_exampledb_node0002 = new_host0002
v_exampledb_node0003 = new_host0003
v_exampledb_node0004 = new_host0004
#v_exampledb_node0001 = 10.0.90.17
#v_exampledb_node0002 = 10.0.90.18
#v_exampledb_node0003 = 10.0.90.19
#v_exampledb_node0004 = 10.0.90.20

[Misc]
; Specifies which tables and/or schemas to copy. For tables, the containing schema defaults to
public.
; !!Mandatory!! objects for replication. The replicate task does not support full database copy
; Use comma-separated list for multiple objects
; (no default)
objects = mytable, myschema, myothertable

; Specifies how Vertica handles objects of the same name when copying schema or tables.
; objectRestoreMode = createOrReplace

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to replicate.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True
```

```
; !!Mandatory!! These settings are all mandatory for replication. None of which have defaults.
dest_dbName = target_db
dest_dbUser = dbadmin
dest_dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between retry attempts, if a failure occurs.
; retryDelay = 1

; Full path to the password configuration file containing database password credentials
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all backup connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bwlimit_backup = 0

; The maximum number of backup TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bwlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is very rarely be needed since dbUser is normally identical to the database administrator.
; dbUser = current_username
```

Backup/Restore To Amazon S3

```
; This sample vbr configuration file shows backup to AWS s3 shared storage
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; Option and values are separated by an equal sign.
; Only arguments marked as '!!Mandatory!!' must be specified explicitly.
; All commented parameters are set to their default value.

; ----- ;
;;; BASIC PARAMETERS ;;;
; ----- ;

[S3]
; This section replaces the [Mapping] section and is required to back up to s3

; !!Mandatory!! S3 bucket name(no default).
s3_backup_path = s3://backup_bucket/database_backup_path/

; !!Mandatory!! directory used to manage locking during a backup (no default). If the directory is
mounted on the initiator host, you should use "[" instead of the local host name. The file system
must support POSIX fcntl flock.
s3_backup_file_system_path = [ ]:/home/dbadmin/backup_locks_dir/
; s3_backup_file_system_path = otherhost.example:/home/dbadmin/backup_locks_dir/

; Specifies SSL encrypted transfer.
; s3_encrypt_transport = True

; Specifies the number of threads for upload/download - backup
; s3_concurrency_backup = 10

; Specifies the number of threads for upload/download - restore
; s3_concurrency_restore = 10

[Misc]
; !!Recommended!! Snapshot name
; Valid values: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
; dbPromptForPassword = True

; ----- ;
;;; ADVANCED PARAMETERS ;;;
; ----- ;

[Misc]
; The temp directory location on all database hosts.
```

```
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl  
lockf locking.  
; tempDir = /tmp/vbr  
  
; Specifies the number of historical backups to retain in addition to the most recent backup.  
; 1 current + n historical backups  
; restorePointLimit = 1  
  
; Full path to the password configuration file  
; Store this file in directory readable only by the dbadmin.  
; (no default)  
; passwordFile = /path/to/vbr/pw.txt  
  
[Database]  
; Vertica user name for vbr to connect to the database.  
; This is rarely needed since dbUser is normally identical to the database administrator  
; dbUser = current_username
```

VBR Configuration File Reference

The configuration file options are grouped into sections within the configuration file:

- [\[Misc\] Miscellaneous Settings](#)
- [\[Database\] Database Access Settings](#)
- [\[Transmission\] Data Transmission During Backup Process](#)
- [\[Mapping\]](#)
- [\[NodeMapping\]](#)

Related Terms

- database administrator
- [NODES](#) system table

Related Tasks

- [Configuring Backup Hosts](#)
- [Creating vbr Configuration Files](#)
- [Configuring the Hard-Link Local Parameter](#)

[Misc] Miscellaneous Settings

This section collects basic settings, including the name and location of your backup. The section also indicates whether you are keeping more than a single backup file, as specified by the `restorePointLimit` parameter.

Parameter	Default	Configuration Setting and Description
<code>snapshotName</code>	<code>snapshotName</code>	<p>Specifies the name of the directory vbr creates for the full or object-level backup. This directory appears beneath a top-level directory named for the node being backed up.</p> <p>Valid Values:</p> <ul style="list-style-type: none">• a–z• A – Z• 0 – 9• Hyphen (-)• Underscore (_)
<code>tempDir</code>	<code>/tmp/vbr</code>	<p>Specifies an absolute path to a temporary storage area on the cluster nodes. The <code>tmp</code> path must be the same on all nodes in the cluster. The vbr utility uses this directory as a temporary location while it is copying files from the source cluster node to the destination backup location.</p>

Parameter	Default	Configuration Setting and Description
		<p>Vertica also writes backup logs to this location. The vbr utility uses this directory as a temporary location for log files, lock files and other bookkeeping information during the execution of a task.</p> <p>Do not specify the same location as your database's data or catalog directory. Any unexpected files or directories in your data or catalog location can cause errors during database start or restore.</p> <p>The file system at this location must support <code>fcntl lockf</code> (POSIX) file locking.</p>
restorePointLimit	1	<p>Specifies the number of historical backups to retain in addition to the most recent backup. For example, if you set <code>restorePointLimit=3</code>, Vertica saves three historical backups, in addition to the most recent backup, for a total of four backups. By default, Vertica maintains a current backup and one historical backup. Saving multiple backups lets you back up incrementally. Enter a positive integer.</p> <p>Saves multiple backups to the same location, which are shared through hard links. In such cases, <code>listbackup</code> displays the common backup prefix but indicates unique time and date suffixes:</p> <pre>my_archive20111111_205841</pre>
objects	None	<p>Specifies whether vbr creates a full or object-level backup. If you do not specify any objects, vbr creates a full backup. Otherwise, specify the object names (schemas or tables) to include in a</p>

Parameter	Default	Configuration Setting and Description
		<p>backup. To enter more than one object, enter multiple names in a comma-separated list.</p> <p>Enter table names in the form <code>schema.objectname</code>. For example, to make backups of the table <code>customers</code> from the schema <code>finance</code>, enter <code>finance.customers</code>. If a public table and a schema have the same name, <code>vbr</code> backs up only the schema. Use the <code>schema.objectname</code> convention to avoid confusion.</p> <p>Object names can include UTF-8 alphanumeric characters. Object names cannot include escape characters, single quote (') or double quote (") characters.</p> <p>To use non-alphanumeric characters, use a backslash (\) followed by a hex value. For instance, if the table name is a <code>my table</code> (<code>my</code> followed by a space character, then <code>table</code>), enter the object name as follows:</p> <pre>objects=my\20table</pre> <p>This parameter also identifies objects that you want to replicate to an alternate cluster.</p>
objectRestoreMode	createOrReplace	<p>Specifies how Vertica handles objects of the same name when restoring schema or table backups.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • createOrReplace • create • coexist

Parameter	Default	Configuration Setting and Description
		For descriptions of these settings, refer to Creating vbr Configuration Files .
retryCount	2	Specifies the number of backup attempts to complete execution after an error occurs. If the backup fails after exceeding the number of retry attempts, the utility reports an error and stops processing.
retryDelay	1	Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
passwordFile	None	Specifies the file name of the password configuration file .
enableFreeSpaceCheck	True	When enabled, Vertica confirms that the specified backup locations contain sufficient free space and inodes to allow a successful backup. If a backup location has insufficient resources, Vertica displays an error message and cancels the backup. If Vertica cannot determine the amount of available space or number of inodes in the backupDir, it displays a warning and continues with the backup. If you do not include this setting in your configuration file, Vertica performs the space check by default.
SnapshotEpochLagFailureThreshold	3600	When performing a backup, replication, or copycluster, specifies the maximum acceptable difference, in seconds, between the current epoch and the backup epoch. If the time between the current epoch and the backup epoch exceeds the value specified in this parameter, Vertica displays an error message. A value of 0 disables this check. If you do not include this setting in your

Parameter	Default	Configuration Setting and Description
		configuration file, Vertica performs this check using the default value. Backups do not contain information committed after the backup epoch.

[Database] Database Access Settings

Sets options for accessing the database.

Parameter	Default	Description
dbName	N/A	Specifies the name of the database to back up. If you do not supply a database name, the vbr utility selects the current database to back up. Micro Focus recommends that you provide a database name.
dbUser	Current user name	Identifies the Vertica user used for database operations performed by vbr. In the case of the replicate task, this user is the source database user. The vbr utility obtains this information automatically as the current user of the person who invoked the <code>--setupconfig</code> command. You must be logged on as the database administrator to back up the database. The password, if you choose to save it, is stored in the Password Configuration File .
dbPromptForPassword	True	Controls whether the utility prompts for a password. If you set this parameter to <code>False</code> (indicating no prompt at run time), then you must also enter the database administrator password in the <code>dbPassword</code> parameter. If you do not supply a password in the configuration file, the utility prompts for one at run time.

Vertica uses destination database parameters only to replicate objects to alternate clusters. By default, these parameters do not exist in your vbr configuration files. You must manually [edit your database configuration](#) file to add them.

Parameter	Default	Description
dest_dbName	N/A	Specifies the name of the destination database.
dest_dbUser	N/A	Identifies the Vertica user name in the destination database to be used for loading the replicated data. This Vertica user must have default super user privileges.
dest_dbPromptForPassword	N/A	Controls whether the utility prompts for a password for the destination database. If you set this parameter to <code>False</code> (indicating no prompt at run time), then you must also enter the destination database administrator password in the <code>dest_dbPassword</code> parameter. If you do not supply a password in the configuration file, the utility prompts for one at run time.

[Transmission] Data Transmission

Sets options for transmitting the data when using backup hosts.

Parameter	Default	Description
encrypt	False	Controls whether the transmitted data is encrypted while it is being copied to the target backup location. Choose this option if you are performing a backup over an untrusted network (for example, backing up to a remote host across the Internet). Note: Encrypting data transmission causes significant processing overhead and slows transfer. One of the processor cores of each database node is consumed during the encryption process. Use this option only if you are concerned about the security of the network used when transmitting backup data.
checksum	False	Controls whether the <code>vbr</code> utility has <code>rsync</code> use the <code>md5</code> checksum to determine whether files are identical before and after network transmission. By default, <code>rsync</code> does not perform checksum. Instead, it

Parameter	Default	Description
		<p>performs minimal file checking, confirming that the file size and time of last modification are identical before and after transmission.</p> <p>Note: Calculating checksum values increases processor usage during the backup process.</p>
port_rsync	50000	Changes the default port number for the rsync protocol. Change this value if the default rsync port is in use on your cluster, or you need rsync to use another port to avoid a firewall restriction.
total_bwlimit_backup	0	The total bandwidth limit in KBps for backup connections. Vertica distributes this bandwidth evenly among the number of connections set in concurrency_backup. The default value of 0 allows unlimited bandwidth.
concurrency_backup	1	The maximum number of backup TCP rsync connection threads per node. To improve local and remote backup, replication, and copy cluster performance, you can increase the number of threads available to perform backups. Increasing the number of threads allocates more CPU resources to the backup task and can, for remote backups, increase the amount of bandwidth used. The optimal value for this setting depends greatly on your specific configuration and requirements. Values higher than 16 produce no additional benefit.
total_bwlimit_restore	0	The total bandwidth limit in KBps for restore connections. Vertica distributes this bandwidth evenly among the number of connections set in concurrency_restore. The default value of 0 allows unlimited bandwidth.
concurrency_restore	1	<p>The maximum number of restore TCP rsync connections per node.</p> <p>The maximum number of restore TCP rsync connection threads per node. To improve local and</p>

Parameter	Default	Description
		remote restore, replication, and copy cluster performance, you can increase the number of threads available to perform restores. Increasing the number of threads allocates more CPU resources to the restore task and can, for restores of remote backups, increase the amount of bandwidth used. The optimal value for this setting depends greatly on your specific configuration and requirements. Values higher than 16 produce no additional benefit.
serviceAccessUser	None	The user name used for simple authentication of rsync connections. This user is neither a Linux nor Vertica user name. It is simply an arbitrary identifier used by the rsync protocol. If you do not provide a user name, Vertica leaves rsync running without authentication, creating a potential security risk. If you choose to save the password, Vertica stores it in the Password Configuration File .
hardLinkLocal	False	Creates a full- or object-level backup using hard file links on the local file system, rather than copying database files to a remote backup host. Add this configuration parameter manually to the Transaction section of the configuration file, as described in Configuring the Hard-Link Local Parameter .
port_ssh_backup	22	<p>Overrides the default SSH port setting (22) for the backup hosts. Enter the required SSH port for your site.</p> <p>Changing the default SSH port is supported only when using the backup and restore tasks. Using a non-default SSH port with the copyCluster task is not supported.</p> <p>NOTE: This parameter is not included in the configuration file automatically. For more information, see Configuring Backup Hosts.</p>

[Mapping]

You have one [Mapping] section for all of the nodes in your database cluster. The section must appear in your configuration file because it specifies asl database nodes being included in the backup. It also includes the backup host and directory for each node. If you have objects being replicated to an alternate database, the [Mapping] section also maps the target database nodes to the source database backup locations.

- If you edit an existing configuration file to add a Mapping in the current style, you must combine information from all existing Mappings into the new section.
- Alternatively, you can use vbr with the --setupconfig option to generate a new configuration file, as described in [Creating vbr Configuration Files](#).

Note: The [S3] and [Mapping] configuration sections are mutually exclusive. If you include both, your backup fails with the error message "Config has conflicting sections (Mapping, S3), specify only one of them."

Parameter	Default	Description
backupHost	None	<p>Indicates the target host name or IP address on which to store this node's backup. The backupHost name is different from dbNode. The copycluster command uses this value to identify the target database node host name.</p> <p>Performance Consideration:</p> <p>Although supported, backups to an NFS host may have poor performance, particularly on networks shared with rsync operations.</p>
backupDir	None	<p>Identifies the full path to the directory on the backup host or node where the backup will be stored.</p> <p>Directory Requirements:</p> <ul style="list-style-type: none"> • Must already exist when you run the utility with the --task backup option • Must be writable by the user account used to run the backup utility.

Parameter	Default	Description
		<ul style="list-style-type: none">• Must be unique to the database you are backing up. Multiple databases cannot share the same backup directory.• The file system at this location must supportfcntl lockf file locking.
dbNode	None	<p>The name of the database node, as recognized by Vertica. This value is not the node's host name, but rather the name Vertica uses internally to identify the node, usually in the form of:</p> <pre>v_node00xx</pre> <p>To find database node names in your cluster, query the node_name column of the NODES system table.</p>

Map to the localhost

Vertica vbr does not support the special localhost name as a backup host. To backup a database node to its own disk, use empty square brackets for the hostname in the [Mapping] section of the configuration file.

```
[Mapping]
NodeName = []:/backup/path
```

Your mapping section should resemble this example:

```
[Mapping]
v_node0001 = []:/scratch_drive/archive/backupdir
v_node0002 = []:/scratch_drive/archive/backupdir
v_node0003 = []:/scratch_drive/archive/backupdir
```

Map to the Same Database

The following example shows how you can specify a Mapping section that indicates a single node to be backed up (v_vmart_node0001). The node is assigned to the backup host (v_srv01), and the backup directory (/home/dbadmin/backups). Although you are backing up

a single node cluster, and the backup host and the database node are the same system, you specify them differently.

Specify the backup host and directory, using a colon (:) as a separator:

```
[Mapping]
v_vmart_node0001 = srv01:/home/dbadmin/backups
```

Although the configuration file `[Mapping]` section no longer uses named parameters, you still use the elements of the simplified format continue to represent the following parameters:

```
dbNode = backupHost:backupDir
```

Map to an Alternate Database

Before you can replicate objects to an alternate database, you must also create a [\[NodeMapping\]](#) section in your vbr configuration file. The `NodeMapping` section points source nodes to their target database nodes.

Restore an alternate database, by adding mapping information in the following form:

```
[Mapping]
targetNode: sourceDBNode_backuphost:sourceDB_backuppath
```

Your mapping section should resemble this example:

```
[Mapping]
v_sec_node0001 = pri_bsrv01:/archive/backup
v_sec_node0002 = pri_bsrv02:/archive/backup
v_sec_node0003 = pri_bsrv03:/archive/backup
```

[NodeMapping]

Vertica uses the node mapping section exclusively to restore a full backup from one database to another, different, database. Be sure to update the [\[Mapping\]](#) section of your configuration file to point your target Vertica nodes to their source backup locations.

Important: Vertica does not automatically generate the [NodeMapping] section of the vbr configuration file. You must edit the file manually to add a node mapping section. Refer to [Full Backup and Restore to an Alternate Cluster](#) as an example.

Use the following form to specify node mapping:

```
source_vertica_node = target_vertica_host
```

For example, you can use the following mapping to restore content from one 4-node database to an alternate 4-node database.

```
[NodeMapping]
v_exampledb_node0001 = new_host0001
v_exampledb_node0002 = new_host0002
v_exampledb_node0003 = new_host0003
v_exampledb_node0004 = new_host0004
```

[S3]

Sets options for storing backup data on Amazon S3.

Note: The [S3] and [Mapping] configuration sections are mutually exclusive. If you include both, your backup fails with the error message "Config has conflicting sections (Mapping, S3), specify only one of them."

Parameter	Default	Description
s3_backup_file_system_path		<p>Specifies the host and path that you are using to handle file locking during the backup process. Vertica must be able to create a passwordless ssh connection to the location that you specify here.</p> <p>To use a local NFS file system, specify a value of:</p> <pre>s3_backup_file_system_path = [:path]</pre> <p>To use an EC2 instance, specify a value of:</p> <pre>s3_backup_file_system_path = [host_name]:path</pre>
s3_backup_path		<p>Specifies the S3 bucket name and backup path for the backup to S3. When you backup to S3, all nodes back up to same S3 bucket. You must create the backup location on S3</p>

Parameter	Default	Description
		before performing a backup. This value takes the following form: <pre>s3_backup_path = s3://backup_bucket/database_backup_path/</pre>
s3_encrypt_transport	True	When True, uses SSL encryption to encrypt moving between your Vertica cluster and your S3 instance. If you are backing up or restoring from an Amazon EC2 cluster, you must set this parameter to True.
s3_concurrency_backup	10	The maximum number of concurrent backup threads for backup to S3.
s3_concurrency_restore	10	The maximum number of concurrent restore threads for restoring from S3.

Backup and Restore Utility Reference

This section provides reference information about both the `vbr` utility commands, and its associated configuration file parameters.

VBR Utility Reference

Allows you to back up and restore either the full database, or one or more schema and table objects of interest. You can also copy a cluster and list backups you created previously.

The utility is located in the Vertica binary directory (`/opt/vertica/bin/vbr` on most installations).

Syntax

```
    /opt/vertica/bin/vbr { command }  
... [ --archive timestamp ]  
... [ --config-file file ]  
... [ --debug level ]  
... [ --nodes node1 [, noden, ...] ]  
... [ --showconfig ]
```

Where *command* is one of the following:

Full Command	Short Command	Description
<code>--help</code>	<code>-h</code>	Shows a brief usage guide for the command.
<code>--showconfig</code>		Displays the current configuration settings.
<code>--setupconfig</code>		Asks a series of questions and generates a configuration file.
<code>--task {backup 7.2_upgrade collect-garbage copycluster full-check init listbackup quick-check quick-repair remove replicate</code>	<code>-t</code>	Performs the specified task: <ul style="list-style-type: none">7.2_upgrade — Upgrades an existing 7.1 or earlier backup to the manifest-based backup introduced in version 7.2.backup — Creates a full database or object-level backup, depending on configuration file

Full Command	Short Command	Description
restore }		<p>specification.</p> <ul style="list-style-type: none"> • <code>collect-garbage</code> — Rebuilds the backup manifest and deletes any unreferenced objects in the backup location. • <code>copycluster</code> — Copies the database to another Vertica cluster. • <code>full-check</code> — Verifies all objects listed in the backup manifest against file system metadata, outputting missing and unreferenced objects. • <code>init</code> — Creates a new backup directory, or prepares an existing one, for use and creates necessary backup manifests. You must perform this task before the first time you create a backup in a directory. • <code>listbackup</code> — Displays the existing backups associated with the configuration file you supply. View this display to get the name of a backup that you want to restore. • <code>quick-check</code> Confirms that all backed-up objects appear in the backup manifest. Outputs any discrepancies between objects in the backup location and objects listed in the backup manifest. • <code>quick-repair</code> — Builds a replacement backup manifest, based on storage locations and objects. • <code>remove</code> — Removes the specified backup or restore point. • <code>replicate</code>— Copies objects from one cluster to an alternate cluster. • <code>restore</code> — Restores a full or object-level database backup. Requires the configuration file that created the backup.

Parameters

Parameter	Description
<code>--archive <i>timestamp</i></code>	Used with the <code>--task restore</code> and <code>--task remove</code> commands. Specifies the timestamp of the backup to restore or remove: > <code>vbr --task restore --config-file myconfig.ini --archive=20160115_182640</code>
<code>-c <i>file</i></code> <code>--config-file <i>file</i></code>	Indicates the configuration file to use as an absolute or relative path to the location from which you start the backup utility. If no file exists, an error occurs and the utility cannot continue.
<code>--nodes <i>node1[,...]</i></code>	Specifies any nodes, in a comma-separated list, on which to perform a vbr task. The nodes listed match the names in the Mapping section of the configuration file. Caution: Do not try to restore the entire database cluster from a partial database backup created from a subset of the nodes. Data loss could result.
<code>--debug <i>level</i></code>	Specifies the level of debugging messages (from 0 to 3) that the vbr utility provides. Level 3 indicates verbose, while level 0, the default, indicates no messages.
<code>--report-file <i>path/filename</i></code>	Optional. Outputs a delimited JSON file that describes the results of the associated full backup integrity check or garbage collection task.
<code>--restore-objects <i>objects</i></code>	Specifies the individual objects to restore from a full or object-level backup.
<code>--s3-force-init</code>	Used with the <code>--task init</code> command. Forces the <code>init</code> task to succeed on S3 storage targets when there is an identity/lock file mismatch. For more information, refer to Creating Backups on Amazon S3 .
<code>--showconfig</code>	The configuration values being used to perform the task, displayed in raw JSON format before vbr starts begins.

Parameter	Description
<code>--list-all</code>	Used with the <code>--task listbackup</code> command. Displays a list of all backups stored on the hosts and paths listed in the specified configuration file.
<code>--json</code>	Used with the <code>--task listbackup</code> command. Displays a JSON delimited list of all backups stored on the hosts and paths listed in the specified configuration file.
<code>--list-output-file <i>path/filename</i></code>	Used with the <code>--task listbackup</code> command. Outputs a file containing a JSON delimited list of all backups stored on the hosts and paths listed in the specified configuration file.

See Also

- [VBR Configuration File Reference](#)

Related Tasks

- [Configuring the Backup Script](#)
- [Restoring Full Database Backups](#)
- [Restoring Object-Level Backups](#)

Password Configuration File Reference

The `vbr` utility automatically creates the password configuration file to store any saved passwords. Only the database administrator or members of that user's permission group can view the contents of the file. If you alter the file permissions from their default values (xx0), Vertica backup or restore actions fail with an error message.

Vertica creates the password file, even if you do not save any passwords. In this case, the file exists but remains empty. Vertica stores this file in the same location that you choose to save your backup and restore configuration file.

[Passwords] Password Settings

Parameter	Default	Description
dbPassword	None	<p>Identifies the database administrator's password. If you set <code>dbPromptForPassword</code> to <code>False</code>, enter a password. Doing so prevents you from being prompted at run time. You do not need to specify additional settings.</p> <p>Empty strings:</p> <p>You cannot enter an empty string for the <code>dbPassword</code> in the configuration file. Micro Focus does not recommend using an empty string as a superuser password. If you do so, you must:</p> <ol style="list-style-type: none">1. Set the <code>dbPromptForPassword</code> parameter to <code>True</code>.2. Leave the <code>dbPassword</code> option blank.3. Enter the empty string at the prompt each time you run the backup utility.
serviceAccessPass	None	Identifies the password for the <code>rsync</code> user account.
dest_dbPassword	None	The password for the <code>dest_dbuser</code> Vertica account. Vertica uses this value for replication tasks only.

When to Back up the Database

In addition to any guidelines established by your enterprise, Micro Focus recommends that you back up your database in the following circumstances:

Before:

- You upgrade Vertica to another release.
- You drop a partition.

- You add, remove, or replace nodes in your database cluster.

After:

- You load a large volume of data.
- You add, remove, or replace nodes in your database cluster.
- You recover a cluster from a crash.

If:

The epoch in the latest backup is earlier than the current ancient history mark.

Note: Always create a new full backup after adding, removing, or replacing nodes. When you restore a full database backup, you must restore to a cluster that is identical to the one on which you created the backup.

Ideally, schedule ongoing backups to back up your data. You can run the Vertica `vbr` from a cron job or other task scheduler.

Related Terms

- Ancient history mark (AHM).

Configuring Backup Hosts

The `vbr` utility lets you back up your database to one or more hosts (known as *backup hosts*), that can be outside of your database cluster.

You can use one or more backup hosts or a single S3 bucket to back up your database. Use the `vbr` configuration file to specify which backup host each node in your cluster should use.

Before you back up to hosts outside of the local cluster, configure the target backup locations to work with the `vbr` utility. The backup hosts you use must:

- [Have sufficient backup disk space.](#)
- Be accessible from your database cluster through SSH.
- Have passwordless SSH access for the Database Administrator account.

- Set `AllowTcpForwarding = Yes` in your `sshd_config` file. If TCP forwarding is not enabled, backups fail with the following message:

"Errors connecting to remote hosts: Check SSH settings, and that the same Vertica version is installed on all nodes. "

- Have either the Vertica rpm or Python 2.7 and rsync 3.0.5 or later installed.
- If you are using a stateful firewall, configure your `tcp_keepalive_time` and `tcp_keepalive_intvl` `sysctl` settings to use values less than your firewall timeout value.

Creating Configuration Files for Backup Hosts

Create separate configuration files for full or object-level backups, using distinct names for each configuration file. Also, use the same node, backup host, and directory location pairs. Specify different backup directory locations for each database.

Note: For optimal network performance when creating a backup, Micro Focus recommends that you give each node in the cluster its own dedicated backup host.

Preparing Backup Host Directories

Before `vbr` can back up a database, you must prepare the target backup directory. Run `vbr` with a task type of `init` to create the necessary manifests for the backup process. You need to perform the `init` process only once. After that, Vertica maintains the manifests automatically.

Estimating Backup Host Disk Requirements

Wherever you plan to save data backups, consider the disk requirements for incremental backups at your site. Also, if you use more than one archive, multiple archives potentially require more disk space. Micro Focus recommends that each backup host have space for at least twice the database node footprint size. Follow this recommendation regardless of the specifics of your site's backup schedule and retention requirements.

To estimate the database size from the `used_bytes` column of the `storage_containers` system table:

```
VMart=> select sum(used_bytes) from storage_containers where node_name='v_mydb_node0001';
total_size
-----
      302135743
(1 row)
```

If your site uses multiple backup host locations, you can estimate the database size requirements per node. Use a query, such as the following, substituting a backup host name for *node_name*:

```
select node_name,sum(used_bytes) as size_in_bytes from v_monitor.storage_containers group by node_name;
```

Making Backup Hosts Accessible

You must verify that any firewalls between the source database nodes and the target backup hosts allow connections for SSH and rsync on port 50000.

The backup hosts must be running identical versions of rsync and Python as those supplied in the Vertica installation package.

Setting Up Passwordless SSH Access

To access a backup host, the database administrator must meet two requirements to run the vbr utility:

- Have an account on each backup host, with write permissions to the backup directory.
- Have passwordless SSH access from each database cluster host to the corresponding backup host.

How you fulfill these requirements depends on your platform and infrastructure.

SSH access among the backup hosts and access from the backup host to the database node is not necessary.

If your site does not use a centralized login system (such as LDAP), you can usually add a user with the `useradd` command or through a GUI administration tool. See the documentation for your Linux distribution for details.

If your platform supports it, you can enable passwordless SSH logins using the `ssh-copy-id` command to copy a database administrator's SSH identity file to the backup location from one of your database nodes. For example, to copy the SSH identity file from a node to a backup host named `backup01`:

```
> ssh-copy-id -i dbadmin@backup01|  
Password:
```

Try logging into the machine with "ssh 'dbadmin@backup01' ". Then, check the contents of the `~/.ssh/authorized_keysfile` to verify that you have not added extra keys that you did not intend to include.

```
> ssh backup01  
Last login: Mon May 23 11:44:23 2011 from host01
```

Repeat the steps to copy a database administrator's SSH identity to all backup hosts you use to back up your database.

After copying a database administrator's SSH identity, you should be able to log in to the backup host from any of the nodes in the cluster. You are not prompted for a password.

Increasing the SSH Maximum Connection Settings for a Backup Host

If your configuration requires backing up multiple nodes to one backup host (n:1), increase the number of concurrent SSH connections to the SSH daemon (`sshd`). By default, the number of concurrent SSH connections on each host is 10, as set in the `sshd_config` file with the `MaxStartups` keyword. The `MaxStartups` value for each backup host should be greater than the total number of hosts being backed up to this backup host.

To increase the `MaxStartups` value:

1. Log on as root to access the config file.
2. Open the SSH configuration file (`/etc/ssh/sshd_config`) in a text editor.
3. If the `MaxStartups` line is commented out with `#`, delete that character. Replace the current value, which may be a single integer or three integers like `10:30:60` with the new value. For example, to back up a 50 node cluster to one machine, use a value of 60:

```
MaxStartups 60
```

For more information on configuring `MaxStartups`, [refer to the man page for that parameter](#).

4. Save the file.
5. Reload the file using the following command:

```
sudo /etc/init.d/sshd reload
```

If you are using Red Hat 7/CentOS 7, use the following command instead: `sudo /bin/systemctl reload sshd.service`

6. Exit from root.

See Also

- [Backup Configuration Options](#)
- [Creating Backups on Amazon S3](#)

Related Tasks

- [Enable Secure Shell \(SSH\) Logins.](#)
- [Configuring Advanced VBR Options](#)

Types of Backups

The `vbr` utility supports several kinds of backups:

- [Full backups](#)
- [Object-level backups](#)
- [Hard-link local backups](#)

You can refer to full or object-level backups by a user-defined descriptive name, such as `FullDBSnap`, `Schema1Bak`, or `Table1Bak`.

Full Backups

A *full backup* is a complete copy of the database catalog, its schemas, tables, and other objects. This type of backup provides a consistent image of the database at the time the backup occurred. You can use a full backup for disaster recovery to restore a damaged or incomplete database. You can also [restore individual objects from a full backup](#).

When a full backup exists, `vbr` creates incremental backups as successive backups following the full-backup. Incremental backups contain new or changed data since the last full, or incremental, backup occurred.

Archives contain a collection of same-name backups. Each archive can have a different retention policy. For example, suppose that `TBak` is the name of an object-level backup of table `T`, and you create a daily backup each week. These seven backups become part of the `TBak` archive. Keeping a backup archive lets you revert back to any one of the saved backups.

Full and object-level backups reside on *backup hosts*, the computer systems on which backups and archives are stored.

Vertica saves backups in a specific *backup location*, the directory on a backup host. This location can contain multiple backups, including associated archives. The backups are also compatible, allowing you to restore any objects from a full database backup.

Object-Level Backups

An *object-level backup* consists of one or more schemas or tables or a group of such objects. The conglomerate parts of the object-level backup do not contain the entire database. When an object-level backup exists, you can restore all of its contents or [individual objects](#).

Note: Vertica does not support object level backups on Hadoop Distributed File System (HDFS) storage.

Object-level backups contain the following object types:

Selective objects	Objects you choose to be part of an object-level backup. For example, if you specify tables <code>T1</code> and <code>T2</code> to include in an object-level backup, they are the selected objects.
Dependent objects	Objects that must be included as part of an object-level backup, due to dependencies. Suppose you want to create an object-level backup that includes a table with a foreign key. To do so, table constraints require that you include the primary key table, and <code>vbr</code> enforces this requirement. Projections anchored on a table in the selected objects are also <i>dependent objects</i> .
Principal objects	The objects on which both selected and dependent objects depend are called <i>principal objects</i> . For example, each table and projection has an owner, and each is a <i>principal object</i> .

Vertica 6.x and later provides support for object-level backups.

Hard Link Local Backups

A *hard-link local backup* can be a full- or object-level backup. It consists of a complete copy of the database catalog, and a set of hard file links to corresponding data files. You must save a hard-link local backup on the file system used by the catalog and database files.

Backup Contents

Backups contain all committed data for the backed up objects as of the start time of the backup. Backups do not contain uncommitted data or data committed during the backup. Backups do not delay mergeout or load activity.

Related Tasks

- [Configuring Backup Hosts](#)

Creating Backups on Amazon S3

Vertica supports the creation of backups on Amazon S3 Standard cloud storage. You can create these backups from your local cluster or from Amazon EC2 virtual servers.

Note: Vertica supports backup and restore from S3. Copycluster and replication to S3 targets is not supported.

Creating an S3 Configuration File

To backup to Amazon S3, you must add an [S3] section to your backup configuration file. For more information, refer to [\[S3\]](#). Vertica also provides a [sample S3 configuration file](#) that you can copy and edit.

Configuring Amazon S3 Storage for Backup

Vertica supports using Amazon S3 cloud storage as a backup location. As with all Vertica backups, Vertica creates incremental backups, meaning each subsequent backup contains only changes that have occurred since you created the first backup. As with any storage location, you must initialize an S3 storage location with the `vbr task init`.

Because S3 storage does not support file locking, Vertica uses either your local file system or an Amazon EC2 file system to handle file locks during a backup. You identify this location using the `s3_backup_file_system_path` [parameter](#) in your `vbr` configuration file. During a backup, Vertica creates a locked identity file on your local or EC2 instance, and a duplicate file in your S3 backup location. As long as the files match, Vertica proceeds with the backup, releasing the lock once the backup is complete. As long as the files remain identical, you can use the S3 location for backup and restore tasks.

If the files in your locking location become out of sync with the files in your backup location, backup and restore tasks fail with an error message. You can resolve locking inconsistencies by rerunning the `init` task with the `--s3-force-init` parameter.

A typical S3 locking file reset command takes the following form:

```
/opt/vertica/bin/vbr --task init --s3-force-init -c filename.ini
```

Note: If a backup fails, confirm that your Vertica cluster has permission to access your S3 storage location.

Configuring EC2 Authentication for Amazon S3

If you are backing to S3 from an EC2-based cluster, you must provide authentication to your S3 host. Regardless of the authentication type you choose, your credentials do not leave your EC2 cluster. Vertica supports the following authentication types:

- AWS credential file
- IAM role
- Environmental variables

AWS credential file - You can manually create a configuration file on your EC2 initiator host at `~/.aws/credentials`.

```
[default]  
aws_access_key_id = YOUR_ACCESS_KEY  
aws_secret_access_key = YOUR_SECRET_KEY
```

For more information on credential files, refer to [Amazon's Web Services documentation](#).

IAM role - Create an AWS IAM role and grant that role permission to access your EC2 cluster and S3 resources. For more information, refer to [Amazon's Web Services documentation](#).

Environmental variables - (Recommended) Amazon Web Services provides the following environmental variables:

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

Use these variables on your initiator to provide authentication to your S3 host. Once your session ends, AWS deletes these variables. For more information, refer to the [AWS documentation](#).

Creating Object-Level Backups

The database administrator can create object-level backups consisting of one or more schemas and tables. Object-level backups are especially useful for multi-tenanted database sites. For example, an international airport could use a multi-tenanted database to represent different airlines in its schemas. Then, tables could maintain various types of information for the airline, including ARRIVALS, DEPARTURES, and PASSENGER information. With such an organization, creating object-level backups of the specific schemas would let you restore by airline tenant, or any other important data segment.

The following configuration file parameters are specific to object-level backups:

- Objects
- objectRestoreMode (used in restore operations only)

For more information about creating configuration files for full or object-level backups, see [Configuring Required VBR Parameters](#).

Note: Apache Kafka uses internal configuration settings to maintain the integrity of your data. When backing up your Kafka data, Micro Focus recommends that you perform a [full database backup](#), rather than an object-level backup.

Preparing Your Backup Directory

Before you can create a backup, you must prepare your backup directory with the [vbr -init task](#).

Invoking vbr Backup

After creating the configuration file specifying which objects to backup, you can create an object-level backup. The following example shows how you can use the `objectbak.ini` command in a configuration file:

```
[dbadmin@v_vmart_node0001 ~]$ vbr --task backup --config-file objectbak.ini
Preparing...
Found Database port: 5433
Copying...
[=====] 100%
All child processes terminated successfully.
Committing changes on all backup sites...
backup done!
[dbadmin@v_vmart_node0001 ~]$
```

Backup Locations and Naming

You can use one top-level backup directory to store both full and object-level backups.

Note: Micro Focus does not recommend concurrent backups. If you must run multiple backups concurrently, use separate backup and temp directories for each. Having separate backup directories detracts from the advantage of sharing data among incremental backups.

To see existing full and object-level backups, see [Viewing Backups](#) in the [See Also](#) section below.

Best Practices for Object-Level Backups

To create one or more object-level backups, create a configuration file specifying the backup location, the object-level backup name, and a list of objects to include (one or more schemas and tables). When creating configuration backup files:

- Create one configuration file for each object-level backup
- Create a different configuration file to create a full database backup
- For best network performance, use one backup host per cluster node
- Use one directory on each backup-node to store successive backups
- For future reference, append the major Vertica version number to the configuration file name (mybackup7x)

Using the same backup host directory location for full and object-level backups results in the backups sharing disk space. Shared disk space makes backups compatible when performing a restore. Each cluster node must also use the same directory location on its designated backup host.

The selected objects of a backup can include one or more schemas or tables, or a combination of both. For example, you can include schema S1 and tables T1 and T2 in an object-level backup. Multiple backups can be combined into a single backup. A schema-level backup can be integrated with a database backup (and a table backup integrated with a schema-level backup, and so on).

Naming Conventions

Give each object-level backup configuration file a distinct and descriptive name. For instance, at an airport terminal, schema-based backup configuration files use a naming convention with an airline prefix, followed by further description, such as:

AIR1_daily_arrivals_backup

AIR2_hourly_arrivals_backup

AIR2_hourly_departures_backup

AIR3_daily_departures_backup

When database and object-based backups exist, you can recover the backup of your choice.

Caution: Do not change object names in an object-level configuration file if a backup already exists. Doing so overwrites the original configuration file, and you cannot restore it from the earlier backup. Instead, create a different configuration file.

Determining Backup Frequency

Micro Focus recommends, as a best practice, that you take frequent backups if database contents diverge in significant ways.

Always take backups after any event that significantly modifies the database, such as performing a rebalance. Mixing many backups with significant differences can weaken data K-safety. For example, taking backups both before and after a rebalance is not a recommended practice in cases where the backups are all part of one archive.

Understanding Object-Level Backup Contents

Object-level backups comprise only the elements necessary to restore the schema or table, including the selected, dependent, and principal objects. An object-level backup includes the following contents:

- Storage: Data files belonging to any specified objects
- Metadata: Including the cluster topology, timestamp, epoch, AHM, and so on
- Catalog snippet: Persistent catalog objects serialized into the principal and dependent objects

Some of the elements that comprise AIR2, for instance, include its parent schema, tables, named sequences, primary key and foreign key constraints, and so on. To create such a backup, `vbr` script saves the objects directly associated with the table. It also saves any dependencies, such as foreign key (FK) tables, and creates an object map from which to restore the backup.

Note: Because the data in local temp tables persists only within a session, local temporary tables are excluded when you create an object-level schema. For global temporary tables, `vbr` stores the table's definition.

Making Changes After an Object-Level Backup

Be aware how changes made after an object-level backup affect subsequent backups. Suppose you create an object-level backup and later drop schemas and tables from the database. In this case, the objects you dropped are also be dropped from subsequent backups. If you do not save an archive of the object backup, such objects could be lost permanently.

Changing a table name after creating a table backup does not persist after restoring the backup. Suppose that, after creating a backup, you drop a user who owns any selected or dependent objects in that backup. In this case, restoring the backup re-creates the object and assigns ownership to the user performing the restore. If the owner of a restored object still exists, that user retains ownership of the restored object.

To restore a dropped table from a backup:

1. Rename the newly created table from t1 to t2.
2. Restore the backup containing t1.
3. Restore t1. Tables t1 and t2 now coexist.

For information on how Vertica handles object overwrites, refer to "Specifying Object Restore Mode" in [Creating vbr Configuration Files](#).

K-safety may increase after an object backup. Restoration of a backup fails if *both* of the following conditions occur:

- An increase in K-safety occurs.
- Any table in the backup has insufficient projections.

Changing Principal and Dependent Objects

If you create a backup and then drop a principal object, restoring the backup restores that principal object. If the owner of the restored object has also been dropped, Vertica assigns the restored object to the current dbadmin.

You can specify how Vertica handles object overwrites in the vbr configuration file. For more information, refer to "Specifying Object Restore Mode" in [Creating vbr Configuration Files](#).

Identity and auto-increment sequences are dependent objects because they cannot exist without their tables. An object-level backup includes such objects, along with the tables on which they depend.

Named sequences are not dependent objects because they exist autonomously. A named sequence remains after you drop the table in which the sequence is used. In this case, the named sequence is a principal object. Thus, you must back up the named sequence with the table. Then, you regenerate it, if it does not already exist when you restore the table. If the sequence does exist, vbr uses it, unmodified. Sequence values could repeat, if you restore the full database and then restore a table backup to a newer epoch.

Considering Constraint References

You must backup all database objects that are related through constraints. For example, suppose you have a schema with tables whose constraints reference only tables in the same schema can be backed up. However, a schema containing a table with an FK/PK constraint on a table in another schema cannot. To back up the second table, you must include the other schema in the list of selected objects.

Configuration Files for Object-Level Backups

The `vbr` utility automatically associates configurations with different backup names but uses the same backup location.

Always create a cluster-wide configuration file and one or more object-level configuration files pointing to the same backup location. Storage between backups is shared, preventing multiple copies of the same data. For object-level backups, using the same backup location causes `vbr` to encounter fewer OID conflict prevention techniques. Avoiding OID conflict prevention results in fewer problems when restoring the backup.

By using cluster and object configuration files with the same backup location, the utility includes additional provisions to ensure that the object-level backups can be used following a full cluster restore. One approach to restoring a full cluster is to use a full database backup to bootstrap the cluster. After the cluster is operational again, you can restore the most recent object-level backups for schemas and tables.

Attempting to restore a full database using an object-level configuration file fails, resulting in this error:

Note:

```
VMart=> /tmp/vbr --config-file=Table2.ini -t restore
Preparing...
Invalid metadata file. Cannot restore.
restore failed!
```

Backup Epochs

Each backup includes the epoch to which its contents can be restored. When `vbr` restores data, Vertica updates to the current epoch.

The `vbr` utility attempts to create an object-level backup five times before an error occurs and the backup fails.

See Also

- [Creating vbr Configuration Files](#)
- [VBR Configuration File Reference](#)
- [Configuring Required VBR Parameters](#)
- [Viewing Backups](#)
- [Types of Backups](#)

Related Tasks

- [Configuring Required VBR Parameters](#)
- [Restoring Object-Level Backups](#)

Creating Hard-Link Local Backups

Before creating a full hard-link local database backup, verify the following:

- Your database is running. All nodes need not be up in a K-safe database for `vbr` to run. However, be aware that any nodes that are DOWN are not backed up.
- The user account that starts the utility (`dbadmin` or other) has write access to the target backup directories.

When you create a full or object-level hard link local backup, that backup contains the following contents.

Backup	Catalog	Database files
Full backup	Full copy	Hard-file links to all database files
Object-level backup	Full copy	Hard-file links for all objects listed in the configuration file, and any of their dependent objects

Run the vbr script from a terminal using the database administrator account from a node in your database cluster. You cannot run the utility as root.

To create a full or object-level backup, enter the following command:

```
> /opt/vertica/bin/vbr --task backup --config fullbak.ini
```

Note: While not required, Micro Focus recommends that you first create a full backup before creating any object-level backups.

Common Errors in Specifying the Hard-Link Local Backup Location

When you specify the hard-link backup location, be sure to avoid these common errors when adding the `hardLinkLocal=True` parameter to the configuration file:

If ...	Then...	Solution
You specify a backup directory on a <i>different</i> node	vbr issues an error message and stops processing the backup.	Change the configuration file to include a backup directory on the same host and file system as the database and catalog files. Then, run the backup utility again.
You specify <i>both</i> of the following: <ul style="list-style-type: none">A backup destination directory on a <i>different</i> file system from the database and catalog files.That backup destination is on the <i>same</i> node.	vbr issues a warning message Performs backup by copying the files on the node from	No action required. No action required.

If ...	Then...	Solution
	one file system to the other.	

Creating Hard-Link Local Backups for Tape Storage

You can use hard-link local backups as a staging mechanism to backup to tape or other forms of storage media. You can also use the hard-link local backup to restore the hard file links to the database files.

The following steps present a simplified approach to saving, and then restoring, hard-link local backups from tape storage:

1. Create a configuration file using a command such as:

```
/opt/vertica/bin/vbr --setupconfig
```

2. Edit the configuration file (`localbak.ini` in this example) to include the `hardLinkLocal=True` parameter in the `[Transmission]` section.
3. Run the backup utility with the configuration file:

```
/opt/vertica/bin/vbr --task backup --config-file localbak.ini
```

4. Copy the hard-link local backup directory with a separate process (not `vbr`) to tape or other external media.
5. If the database becomes corrupted, create the directory structure that existed when you created the hard-link local backup.
6. Transfer the backup files from tape to their original backup directory.
7. Using the configuration file you used to create the hard-link local backup (Step 3), restore the database using the following command:

```
/opt/vertica/bin/vbr --task restore --config-file localbak.ini
```

When you restore from a hard-link local backup (copied from tape), `vbr` creates hard links from the backup files to the database directory, if possible. This approach saves significant disk space and time.

Related Terms

- K-safe
- database administrator

Creating Full Backups

Before you create a database backup, verify the following:

- You have prepared your backup directory with the [vbr -init task](#).
- Your database is running. It is unnecessary for all nodes to be up in a K-safe database. However, any nodes that are DOWN are not backed up.
- All of the backup hosts are up and available.
- The backup host (either on the database cluster or elsewhere) has sufficient disk space to store the backups.
- The user account of the user who starts the utility has write access to the target directories on the host backup location. This user can be the `dbadmin` or another assigned role. However, you cannot run the utility as root.
- Each backup has a unique file name.

Run the `vbr` script from a terminal. Use the database administrator account from an initiator node in your database cluster.

Running `vbr` Without Optional Commands

You can run the `vbr` using only its required commands:

- `--task backup`
- `--config-file config_file`

If your configuration file does not contain the database administrator password, `vbr` prompts you to enter the password. However, it does not display what you type.

The utility requires no further interaction after you invoke it.

To run the `vbr` utility:

Use the `--task backup` and `--config-file filename` directives as shown in this example:

```
[release@qco55srv01:/scratch_b/qa/vertica/QA/VT_Scenario 0]$ vbr -t backup --config $FULLBAK_CONFIG
Starting backup of database VTDB.
Participating nodes: v_vmart_node0001, v_vmart_node0002, v_vmart_node0003, v_vmart_node0004.
Snapshotting database.
Snapshot complete.
Approximate bytes to copy: 2315056043 of 2356089422 total.
[=====] 100%
Copying backup metadata.
Finalizing backup.
Backup complete!
```

By default, no output appears, other than the progress bar. To include additional progress information, use the `--debug` option, with a value between 1–3.

If the utility does not locate the configuration you specify, it searches for one at `opt/vertica/config/vbr.ini`. If no file exists, the backup fails with an error.

Best Practices for Creating Backups

When creating backup configuration files:

- Create separate configuration files to create full- and object-level backups.
- Use the same backup host directory location for both kinds of backups:
 - Because the backups share disk space, they are compatible when performing a restore.
 - Each cluster node must also use the same directory location on its designated backup host.
- For best network performance, use one backup host per cluster node.
- Use one directory on each backup node to store successive backups.
- For future reference, append the major Vertica version number to the configuration file name (`mybackup7x`).

The selected objects of a backup can include one or more schemas or tables, or a combination of both. For example, you can include schema `S1` and tables `T1` and `T2` in an object-level backup. Multiple backups can be combined into a single backup. A schema-level backup can be integrated with a database backup (and a table backup integrated with a schema-level backup, and so on).

Incremental Backups

Each time you back up your database with the same configuration file, `vbr` creates an incremental backup. This incremental backup copies new storage containers, which can include:

- Data that existed the last time you performed a database backup
- New and changed data since the last full backup

You can configure the `restorePointLimit` parameter to increase the number of stored backups.

vbr Process for Deleting Backups

Running the `vbr` utility with the `--task backup` command deletes the oldest backup. This command run whenever the total number that exist exceeds the `restorePointLimit` value in the configuration file. Suppose the `restorePointLimit = 5`, and five archives exist. In this case, running the `vbr --task backup` utility again deletes the backup with the oldest *date_timestamp*, when `vbr` completes the current backup command.

When you invoke `vbr` to create a backup:

1. The utility obtains the value of the `restorePointLimit` parameter value to determine how many backups should exist in the archive.
2. If creating the next backup exceeds the restore point limit, `vbr` deletes the oldest archived backup.
3. `vbr` continues processing and initially creates the backup on the database cluster.
4. When the new backup is complete, `vbr` copies it from the database cluster to the designated backup location.
5. After the new backup is successfully copied to the backup location, `vbr` removes the backup from the database cluster.

See Also

- [Viewing Backups.](#)

Related Terms

- K-safe
- database administrator

Related Tasks

- [Configuring Backup Hosts](#)
- [Creating Object-Level Backups](#)

Interrupting the Backup Utility

To cancel a backup, use **Ctrl+C** or send a SIGINT to the Python process running the backup utility. The utility stops the backup procedure after it has completed copying the data. Canceling a `vbr` backup with **Ctrl+C** immediately closes the session. Uninterrupted incremental backups may run more quickly as a result.

The files generated by an interrupted backup process remain in the target backup location directory. The next backup process picks up where the interrupted process left off.

Backup operations are atomic, so that interrupting a backup operation does not affect the previous backup. Vertica replaces the previous backup only as the very last step of backing up your database.

The `restore` or `copy-cluster` operations overwrite the database catalog directory. Thus, interrupting either of these processes leaves the database unusable until you restart the process and allow it to finish.

Backup and Restore Resource Allocation

Vertica allows administrators to allocate bandwidth and TCP rsync connection resources to backup and restore operations. By default, Vertica allows a single connection and unlimited bandwidth for any backup or restore operation. By changing the default allocations in your vbr configuration file, you can:

- Increase the number of available connections. If you have many Vertica files, more connections can provide a significant performance boost as each connection increases the number of concurrent file transfers.
- Assign a limit to the total amount of bandwidth that a particular operation uses. If your files are fewer and larger, additional bandwidth can provide a performance boost.

For more information, refer to the `total_bwlimit_backup`, `total_bwlimit_restore`, `concurrency_backup`, and `concurrency_restore` settings in [Configuring Advanced VBR Options](#).

Increasing Queue Transport

Each connection creates an additional process that can transport an additional queue of files. When you start a backup or restore operation, Vertica creates a list of all the files requiring transport. As a connection completes a transfer, it takes the next file on the list until the list has been completed. If any file fails to transfer, Vertica assigns that file to another process.

The `retryCount` setting in the vbr configuration file determines the number of times vbr attempts to transfer a file. If the number of failures for a particular file exceeds the limit specified in the `retryCount` setting, the entire operation fails. Vbr then terminates processing.

Bandwidth Limits

Vertica can limit its network bandwidth use through the `total_bwlimit_backup` and `total_bwlimit_restore` data transmission parameters. For more information, refer to [\[Transmission\] Data Transmission](#).

Viewing Backups

You can view backups in any of three ways:

- Use `vbr` to list the backups that reside on the local or remote backup host (requires a configuration file).
- View historical information about backups using the `DATABASE_BACKUPS` systems table. Because the `database_backups` system table contains historical information, it is not updated when you delete the backups.
- Open the `vbr` log file to check the status of a backup. The log file resides on the node where you have run `vbr`, in the directory specified by the [tempDir vbr configuration parameter](#). The default name of this directory is `/tmp/vbr`.

List Backups with `vbr`

To list backups on the backup hosts, use `vbr --task listbackup` with a specific configuration file. The following example shows how you can list backups, using a full backup configuration file, `bak.ini`:

```
dbadmin@doch01 ~]$ vbr --task listbackup --config-file /home/dbadmin/bak.ini
```

The `vbr` output information includes the following:

- `backup` — The name of the generated backup. Vertica names the backup by combining the name of the `vbr` configuration file with a timestamp. Use the timestamp to identify an archive when you perform a restore.
- `backup_type` — The type of the backup, either full or object.
- `epoch` — The epoch when the backup was created.
- `objects` — The objects being backed up. For a full backup, this field is blank.
- `nodes (hosts)` — The hosts that received the backup and the database node names that provided the backups.

- `file_system_type` — The storage location file system of the Vertica hosts that comprise this backup, either Linux or HDFS. For information on backing up HDFS hosts, refer to [Backing Up HDFS Storage Locations](#) .

The following example shows a list of full backups of a three node cluster to a single backup host, bkhost.

```

backup          backup_type epoch  objects  nodes (hosts)
                file_system_type
bak_20160414_134452  full      749      (bkhost), v_vmart_node0003(bkhost) [Linux]
bak_20160413_174544  full      659      (bkhost), v_vmart_node0003(bkhost) [Linux]

```

Note: The `listbackup` task fails if you attempt to view backups on a cluster without a database when the backups were made to local hosts using the [\[\] shortcut](#). `vbr` requires a database to provide the location of the mapped local host.

Viewing All Backups in a Location

You can use the `--list-all` parameter with the `listbackup` task to view a list of all the snapshots stored on the hosts and paths listed in the specified configuration file.

```
dbadmin@doca01 ~]$ vbr --task listbackup --list-all --config-file /home/dbadmin/Nightly.ini
```

The following example shows a `--list-all` task using the configuration file `Nightly.ini`. That configuration file references the hosts `doca01`, `doca02`, and `doca03` and the path `/vertica/backup`. The output shows that these locations contain not just the backups created using `Nightly`, but also backups created using a configuration file called `Weekly.ini`.

```

backup          backup_type epoch  objects  nodes(hosts)
                file_system_type
Weekly_20170508_183249  full      3449      vmart_1(doca01), vmart_2(doca01), vmart_3
(doca01) [Linux]
Weekly_20170508_182816  full      2901      vmart_1(doca01), vmart_2(doca02), vmart_3
(doca03) [Linux]
Weekly_20170508_182754  full      2649      vmart_1(doca01), vmart_2(doca02), vmart_3
(doca03) [Linux]
Nightly_20170508_183034  object    1794      sales_schema vmart_1(doca01), vmart_2(doca02), vmart_3
(doca03) [Linux]
Nightly_20170508_181808  object    1469      sales_schema vmart_1(doca01), vmart_2(doca02), vmart_3
(doca03) [Linux]
Nightly_20171117_193906  object    173       sales_schema vmart_1(doca01), vmart_2(doca02), vmart_3
(doca03) [Linux]

```

You can also use the the `--json` and `--list-output-file` parameters with the `listbackup` task to output the same content in JSON delimited format to a display or to an output file. For more information, refer to [VBR Utility Reference](#).

Query database_backups

Use the following query to list historical information about backups. The `objects` column lists which objects were included in object-level backups. Do not use the `backup_timestamp` value when restoring an archive. Instead, Use the values provided by `vbr --task listbackup`, when restoring an archive.

```
VMart=> SELECT * FROM v_monitor.database_backups;
-[ RECORD 1 ]-----+-----
backup_timestamp | 2013-05-10 14:41:12.673381-04
node_name       | v_vmart_node0003
snapshot_name   | schemabak
backup_epoch    | 174
node_count      | 3
file_system_type | [Linux]
objects         | public, store, online_sales
-[ RECORD 2 ]-----+-----
backup_timestamp | 2013-05-13 11:17:30.913176-04
node_name       | v_vmart_node0003
snapshot_name   | kantibak
backup_epoch    | 175
node_count      | 3
file_system_type | [Linux]
objects         |
-[ RECORD 13 ]----+-----
backup_timestamp | 2013-05-16 07:02:23.721657-04
node_name       | v_vmart_node0003
snapshot_name   | objectbak
backup_epoch    | 180
node_count      | 3
file_system_type | [Linux]
objects         | test, test2
-[ RECORD 14 ]----+-----
backup_timestamp | 2013-05-16 07:19:44.952884-04
node_name       | v_vmart_node0003
snapshot_name   | table1bak
backup_epoch    | 180
node_count      | 3
file_system_type | [Linux]
objects         | test
-[ RECORD 15 ]----+-----
backup_timestamp | 2013-05-16 07:20:18.585076-04
node_name       | v_vmart_node0003
snapshot_name   | table2bak
backup_epoch    | 180
node_count      | 3
file_system_type | [Linux]
objects         | test2
```

Related Topics

- [DATABASE_BACKUPS](#)

Estimating Log File Disk Requirements

When you run the `vbr --setupconfig` command to create the configuration file and configure advanced parameters, one of the parameters is `tempDir`. This parameter specifies the database host location where `vbr` writes its log files and some other temp files (of negligible size). The default location is the `/tmp/vbr` directory on each database host. You can change the default location by specifying a different path in the configuration file.

The temporary storage directory also contains local log files describing the progress, throughput, and any errors encountered for each node. Each time you run `vbr`, the script creates a separate log file, each named with a timestamp. When using default settings, the log file typically uses about 4KB of space per node per backup.

The `vbr` log files are not removed automatically, so you must delete older log files manually, as necessary.

Checking Backup Integrity

Vertica can confirm the integrity of your backup files and the manifest that identifies them. By default, backup integrity checks output their results to the command line.

Perform a Quick Check

A *quick check* gathers all backup metadata from the backup location specified in the configuration file and compares that metadata to the backup manifest. A quick check does not verify the objects themselves. Instead, this task outputs an exceptions list of any discrepancies between objects in the backup location and objects listed in the backup manifest.

Use the following form to perform quick check task:

```
vbr -t quick-check -c configfile.ini
```

For example:

```
vbr -t quick-check -c backupconfig.ini --report-file=logging/quickintegritycheck.json
```

Perform a Full Check

A *full check* verifies all objects listed in the backup manifest against file system metadata. A full check includes the same steps as a quick check. You can include the optional `--report-file` parameter to output results to a delimited JSON file. This task outputs an exceptions list that identifies the following inconsistencies:

- Incomplete restore points
- Damaged restore points
- Missing backup files
- Unreferenced files

Use the following form to perform a full check task:

```
vbr -t full-check -c configfile.ini --report-file=path/filename
```

For example:

```
vbr -t full-check -c backupconfig.ini --report-file=logging/fullintegritycheck.json
```

Repairing Backups

Vertica can reconstruct backup manifests and remove unneeded backup objects. You can include the optional `--report-file` parameter to output results to a delimited JSON file.

Performing a Quick Repair

The `quick-repair` task rebuilds the backup manifest, based on the manifests contained in the backup location.

Use the following form to perform a quick repair task:

```
vbr -t quick-repair -c configfile.ini
```

Performing Garbage Collection

The `collect-garbage` task rebuilds your backup manifest and deletes any backup objects that do not appear in the manifest. You can include the optional `--report-file` parameter to output results to a delimited JSON file.

Use the following form to perform a garbage collection task:

```
vbr -t collect-garbage -c configfile.ini --report-file=path/filename
```

Removing Backups

You can remove existing backups and restore points using the `vbr` utility. When you use the `vbr` task, `remove`, Vertica updates the manifests affected by the removal and maintains their integrity. If the backup archive contains multiple restore points, removing one does not affect the others. When you remove the last restore point, Vertica removes the backup entirely.

Note: Vertica does not support removing backups through the file system.

To remove a backup or a restore point, use the `vbr` task `remove`. Specify this task in the following form:

```
vbr -t remove -c configfile.ini --archive timestamp
```

You can remove multiple restore points using the `archive` parameter. To obtain the timestamp for a particular restore point, [use the `listbackup` task](#).

- To remove multiple restore points, use a comma separator:
`--archive="restore_point1, restore_point2"`
- To remove an inclusive range of restore points, use a colon:
`--archive="restore_point1: restore_point2"`
- To remove all restore points, specify an archive value of `all`:
`--archive="all"`

The following example shows how you can remove a restore point from an existing backup.

```
[dbadmin@doch01 ~]$ vbr -t remove -c backup.ini --archive 20160414_134452  
Removing restore points: 20160414_134452  
Remove complete!
```

Upgrading Pre-7.2 Backups

In version 7.2 and later, Vertica no longer relies on hard links to perform backups. As a result, pre-7.2 backups are not compatible with later Vertica versions. To resolve this issue, `vbr` includes the `vbr` task `7.2_upgrade`. This task copies an existing pre-7.2 backup and creates a 7.2.x-compatible version of it.

Note: Micro Focus recommends that you run this task before performing the first backup of the upgraded database.

To upgrade a backup:

1. Specify the `vbr` task `7.2_upgrade` in the following form:

```
vbr -t 7.2_upgrade --old-config-file outdated-configfile.ini -c new-configfile.ini
```

2. Verify that the `snapshotName` parameter is the same in the old and new configuration files.

The new configuration file assigns new backup locations for the upgraded backup. This approach preserves the existing backup so you can continue to perform incremental backups on the upgraded backup. After the upgrade is complete, Vertica no longer requires the old configuration file.

Using HardFile Link Local Backups

You can use the `vbr` utility `hardLinkLocal` option to create a full- or object-level backups with hard file links on a local database host.

Creating hard-link local backups can provide the following advantages over a remote host backup:

- **Speed** — Hard link local backups are significantly faster than a remote host backup. When backing up, `vbr` does not copy files if the backup directory exists on the same file system as

the database catalog and data directories.

- Reduced network activities — The hard-link local backup minimizes network load because it does not require rsync to copy files to a remote backup host.
- Less disk space — The backup includes a copy of the catalog and hard file links. Therefore, the local backup uses significantly less disk space than a backup with copies of database data files. However, a hard-link local backup saves a full copy of the catalog each time you run `vbr`. Thus, the disk size increases with the catalog size over time.

Hard-link local backups can help you during experimental designs and development cycles. Database designers and developers can create hard-link local object backups of schemas and tables on a regular schedule during design and development phases. If any new developments are unsuccessful, developers can restore one or more objects from the backup.

Note: Running `vbr` does not affect active database applications. The `vbr` utility supports creating backups while concurrently running applications that execute DML statements, including `COPY`, `INSERT`, `UPDATE`, `DELETE`, and `SELECT`.

Planning Hard-Link Local Backups

If you plan to use hard-link local backups as a standard site procedure, design your database and hardware configuration appropriately. Consider storing all of the data files on one file system per node. Such a configuration has the advantage of being automatically set up for hard-link local backups. However, be aware that using one file system per node to support hard-link local backups does preclude the use of external storage locations on separate file systems.

Specifying Backup Directory Locations

The `backupDir` parameter of the configuration file specifies the location of the top-level backup directory. Hard-link local backups require that the backup directory be located on the same Linux file system as the database data and catalog files. The Linux operating system cannot create hard file links to another file system.

Do not create the hard-link local backup directory in a database data storage location. For example, as a best practice, the database data directory should not be at the top level of the file system, as it is in the following example:

```
/home/dbadmin/data/VMart/v_vmart_node0001
```

Instead, Micro Focus recommends adding another subdirectory for data above the database level, such as in this example:

```
/home/dbadmin/data/dbdata/VMart/v_vmart_node0001
```

You can then create the hard-link local backups subdirectory as a peer of the data directory you just created, such as in this example:

```
/home/dbadmin/data/backups  
/home/dbadmin/data/dbdata
```

Hard-Link Local Backups in Disaster Recovery

Hard-link local backups are only as reliable as the disk on which they are stored. If the local disk becomes corrupt, so does the hard link local backup. In this case, you are unable to restore the database from the hard-link local backup because it is also corrupt.

All sites should maintain full backups externally for disaster recovery because hard-link local backups do not actually copy any database files.

Related Tasks

- [Creating Hard-Link Local Backups](#)

Configuring Hard-Link Local Backup Hosts

When specifying the `backupHost` parameter for your hard link local configuration files, use the database host names (or IP addresses) as known to `admintools`. Do not use the node names. Host names (or IP addresses) are what you used when setting up the cluster. Do not use `localhost` for the `backupHost` parameter.

Listing Host Names

To query node names and host names:

```
VMart=> select node_name, host_name from node_resources;  
node_name      | host_name
```

```
-----+-----  
v_vmart_node0001 | 192.168.223.11  
v_vmart_node0002 | 192.168.223.22  
v_vmart_node0003 | 192.168.223.33  
(3 rows)
```

Because you are creating a local backup, use square brackets [] to map the host to the local host. For more information, refer to [\[Mapping\]](#).

```
[Mapping]  
v_vmart_node0001 = [ ]:/home/dbadmin/data/backups  
v_vmart_node0002 = [ ]:/home/dbadmin/data/backups  
v_vmart_node0003 = [ ]:/home/dbadmin/data/backups
```

Restoring Object-Level Backups

To restore an object-level backup to the database from which it was taken, the database must be UP. The vbr configuration file you use for the restore task specifies the backup to restore.

You can specify how Vertica reacts to duplicate objects by configuring the `objectRestoreMode` setting in your vbr configuration file.

Restoring Objects to a Changed Cluster

Unlike restoring from a full database backup, vbr supports restoring object-level backups after adding nodes to the cluster. Any nodes that were not in the cluster when you created the object-level backup do not participate in the restore. You can rebalance your cluster after the restore to distribute data among the new nodes.

You cannot restore an object-level backup after removing nodes, altering node names, or changing IP addresses. Trying to restore an object-level backup after such changes causes vbr to fail and display this message:

```
Preparing...  
Topology changed after backup; cannot restore.  
restore failed!
```

Projection Epoch After Restore

All object-level backup and restore events are treated as DDL events. If a table does not participate in an object-level backup, possibly because a node being down, restoring the

backup affects the projection in the following ways:

- Its epoch is reset to 0.
- It must recover any data that it does not have by comparing epochs and other recovery procedures.

Catalog Locks During Backup Restore

As with other databases, Vertica transactions follow strict locking protocols to maintain data integrity.

When restoring an object-level backup into a cluster that is UP `vbr` begins by copying data and managing storage containers. If necessary, `vbr` splits the containers. This process does not require any database locks.

After completing data copying tasks, `vbr` first requires a table object lock (O-lock) and then a global catalog lock (GCLX).

Sometimes, other database operations, such as DML statements, are in progress when the process attempts to get an O-lock on the table. In such cases, `vbr` is blocked from progress until the DML statement completes and releases the lock. After securing an O-lock first, and then a GCLX lock, `vbr` blocks other operations that require a lock on the same table.

To guarantee catalog consistency, processes can hold a GCLX for a minimal duration. When the restore locks are in effect, any concurrent table modifications are blocked until both locks are released. Database system operations, such as the Tuple Mover (TM) transferring data from memory to disk, are canceled to permit the object-level restore to complete.

Catalog Restore Events

Each object-level backup includes a section of the database catalog, called a *snippet*. A snippet contains the selected objects, their dependent objects, and principal objects. A catalog snippet is similar in structure to the database catalog but consists of a subset representing the object information. Objects being restored can be read from the catalog snippet and used to update both global and local catalogs.

Each object from a restored backup is updated in the catalog. If the object no longer exists, `vbr` drops the object from the catalog. Any dependent objects that are not in the backup are also dropped from the catalog.

The `vbr` utility uses existing dependency verification methods to check the catalog and adds a restore event to the catalog for each restored table. That event also includes the epoch at

which the event occurred. If a node misses the restore table event, it recovers projections anchored on the given table.

Catalog Size Limitations

Object level restores can fail if your catalog size is greater than five percent of the total memory available in the node performing the restore. In this situation, Micro Focus recommends restoring individual objects from the backup. For more information, refer to [Restoring Individual Objects from a Full or Object-Level Backup](#).

See Also

- [Failure Recovery](#)
- [Transactions](#)

Related Tasks

- [Creating Object-Level Backups](#)
- [Restoring Full Database Backups](#)
- [Ownership of Restored Objects](#)

Restoring Full Database Backups

To restore a full database backup, you must verify that:

- The database is DOWN. You cannot restore a full backup when the database is running.
- All of the backup hosts are UP and available.
- The backup directory exists and contains the backups from which to restore the data.

- The cluster to which you are restoring the backup has:
 - The same number of hosts as the one used to create the backup
 - Identical node names
- The target database must already exist on the cluster to which you are restoring data.
 - Database can be completely empty, without any data or schema.
 - The database name must match the name in the backup
 - All of the node names in the database must match the names of the nodes in the configuration file.
- The user performing the restore is the database administrator.

You can use only a full database backup to restore a complete database. If you have saved multiple backup archives, you can restore from either the last backup or a specific archive.

Restoring from a full database backup injects the OIDs from each backup into the restored catalog of the full database backup. The catalog also receives all archives. Additionally, the OID generator and current epoch are set to the current epoch.

Restoring the Most-Recent Backup

To perform a full database restore, the cluster must be DOWN. Usually, when a node or cluster is DOWN, you want to return the cluster to its most-recent state. Doing so requires restoring a full database backup. You can restore any full database backup from the archive by identifying the name in the configuration file.

To restore from the most recent backup, use the configuration file used to create the backup, specifying `vbr` with the `--task restore`. If your [password configuration file](#) does not contain the database superuser password, the utility prompts you to enter it at run time.

The following example shows how you can use the `db.ini` configuration file for restoration:

```
> vbr --task restore --config-file db.ini
Copying...
1871652633 out of 1871652633, 100%
All child processes terminated successfully.
restore done!
```

You can restore a backup only to the database from which it was taken.

Restoring an Archive

If you saved multiple backups, you can specify a specific archive to restore. To list the archives that exist to choose one to restore, use the `vbr --listbackup` task, with a specific configuration file.

To restore from one of several archives:

Log in using the database administrator's account.

Invoke the utility with the `--task restore` command:

- Include the configuration file with which you created the backup.
- Next, add the file name with the `--archive` parameter with the *date_timestamp* suffix of the directory name, which identifies the archive to restore.

For example:

```
> vbr --task restore --config-file fullbak.ini --archive=20121111_205841
```

The `vbr` utility restores the backup.

The `--archive` parameter identifies the archive created on 11-11-2012 (`_archive20121111`), at time 205841 (20:58:41). You need specify only the `_archive` suffix, because the configuration file identifies the backup name of the subdirectory, and the OID identifier indicates the backup is an archive.

See Also

- [Viewing Backups](#)
- [Restoring Individual Objects from a Full or Partial Backup](#)

Related Terms

- database administrator

Full Backup Restore to an Alternate Cluster

Vertica supports restoring a full backup to an alternate cluster.

Requirements for Restoring to an Alternate Cluster

The process is similar to the process for [Restoring Full Database Backups](#), with the following additional requirements.

The destination database must:

- Be DOWN.
- Share the same name as the source database.
- Have the same number of nodes as the source database.
- Have the same names as the source nodes.
- Use the same catalog directory location as the source database.
- Use the same port numbers as the source database.
- Contain the same standby nodes as the backup.

Restore a Full Backup to an Alternate Cluster

1. Copy the [vbr configuration file](#) that you used to create the backup to any node on the destination cluster.
2. If you are using a [stored password](#), copy the password configuration file to the same location as the vbr configuration file.
3. From the destination node, issue a vbr restore command, such as:

```
vbr -t restore -c full.ini
```
4. After the restore has completed, [start the restored database](#).

Restoring Individual Objects from a Full or Object-Level Backup

You can copy individual tables or schemas from any backup that contains those objects without restoring the entire backup. This option is useful if you only need to restore a few objects and want to avoid the overhead of a larger scale restore. Your database must be running, and your nodes must be UP to restore individual objects.

To restore individual objects, the source backup must be from the same version of Vertica as the destination database. Vertica does not support individual object restore across different versions.

When you restore an object, Vertica does not automatically restore any dependent objects. For example, if you restore a schema containing views, Vertica does not automatically restore the tables corresponding to those views.

You can specify how Vertica reacts to duplicate objects by configuring the `objectRestoreMode` setting in your `vbr` configuration file.

Restoring an Object from the Most-Recent Backup

To restore individual objects from the most recent backup, you use the `vbr` utility, just as you did to create the backup. Execute the command using the following form:

```
vbr --task=restore --config-file=filename --restore-objects=objectname,objectname
```

To restore an object, log in using the database administrator's account. You cannot run the utility as root.

To restore from the most recent backup, use the configuration file used to create the backup, specifying the restore `vbr` task. Be sure to include the `--restore-objects` parameter and the names of the objects that you want to restore. Identify tables by fully qualified `schema.name`. To restore multiple objects, use a comma as a delimiter.

If your configuration file does not contain the database superuser password, the `vbr` utility prompts you to enter it at run time.

The following example uses the `db.ini` configuration file, which includes the database administrator's password:

```
> vbr --task restore --config-file=db.ini --restore-objects=saleschema,public.sales_
table,public.customer_info
Preparing...
Found Database port: 5433
Copying...
[=====] 100%
All child processes terminated successfully.
All extract object child processes terminated successfully.
Copying...
[=====] 100%
All child processes terminated successfully.
restore done!
```

Restoring an Object from an Archive

To restore individual objects from a specific archive, you use the same `vbr` utility that you used to create the backup. The command takes the following form:

```
> vbr --task=restore --config-file=filename --archive=date_time --restore-
objects=objectname,objectname
```

Invoke the utility with the `--task restore` command, the configuration file with which you created the backup. Next, add the `--archive` parameter with the *date_timestamp* suffix of the directory name to identify which archive to restore.

The `--archive` parameter identifies the archive subdirectory, including date and time. For example, `_archive20121111_205841` was created on date 11-11-2012 at time 20:58:41. You need specify only the `date_time` suffix because the configuration `.ini` file identifies the backup name of the subdirectory. Additionally, the OID identifier indicates the backup is an archive.

For example:

```
> vbr --task restore --config-file fullbak.ini --archive=20121111_205841 --restore-
objects=salesdb.contacts
```

The `vbr` utility restores the backup.

See Also

- [Monitoring Recovery](#)
- [Viewing Backups](#)

- [Restoring Full Database Backups](#)
- [Restoring Object-Level Backups](#)
- [Ownership of Restored Objects](#)

Restoring Hard-Link Local Backups

Before attempting a hard-link local backup, you must be aware of issues around restoring from this type of backup, whether full or object level.

Process for Restoring Full and Object-Level Hard Link Local Backups

If you have created both full and object-level backups and the database fails, restore the backups in this order:

1. Restore the full database backup.
2. Restore from any object-level backups.

Transferring Backups to and from Remote Storage

When a full hard-link local backup exists, you can use a utility (other than `vbr`) to transfer the backup to another storage media, such as tape. Transferring hard-link local backups to another storage media may copy the data files associated with the hard file links.

You can use a different directory when you return the backup files to the hard-link local backup host. However, you must also change the `backupDir` parameter value in the configuration file before restoring the backup.

Complete the following steps to restore hard link local backups from external media:

1. If the original backup directory no longer exists on one or more local backup host nodes, re-create the directory.

The directory structure into which you restore hard link backup files must be identical to

what existed when the backup was created. For example, if you created hard-link local backups at the following backup directory, you can then re-create that directory structure:

```
/home/dbadmin/backups/localbak
```

2. Copy the backup files to their original backup directory, as specified for each node in the configuration file. For more information, refer to [\[Mapping\]](#).
3. Restore the backup, using one of three options:

- a. To restore the latest version of the backup, move the backup files to the following directory:

```
/home/dbadmin/backups/localbak/node_name/snapshotname
```

- b. To restore a different backup version, move the backup files to this directory:

```
/home/dbadmin/backups/localbak/node_name/snapshotname_archivedate_timestamp
```

4. When the backup files are returned to their original backup directory, use the original configuration file to invoke vbr as follows:

```
>/opt/vertica/bin/vbr --task restore --config-file localbak.ini
```

5. Confirm that the backup has succeeded. Verify that:
 - The physical files are restored from tape into the correct directories.
 - You are using the configuration file that specifies `hardLinkLocal = true`.

Note: You can use a different directory when you return the backup files to the hard-link local backup host. However, you must also change the `backupDir` parameter value in the configuration file before restoring the backup.

Ownership of Restored Objects

When performing a restore, Vertica inserts data into existing database objects. By default, the restore does not affect the ownership, storage policies, or permissions of the restored objects. If, however, the restored object does not currently exist, Vertica re-creates it. In this situation,

the restored object is owned by the user (DBADMIN) performing the restore. Vertica does not restore dependent grants, roles, or client authentications with restored objects.

If the storage policies of a restored object are not valid, vbr applies the default storage policy. Restored storage policies can become invalid due to HDFS storage locations, table incompatibility, and unavailable min-max values at restore time.

Sometimes, Vertica encounters a catalog object that it does not need to restore. When this situation occurs Vertica generates a warning message for that object and the restore continues.

Examples

Suppose you have a full backup, including Schema1, owned by the user, Alice. Schema1 contains Table1, owned by Bob, who eventually passes ownership to Chris. The user dbadmin performs the restore. The following scenarios might occur that affect ownership of the schema:

Scenario 1:

Schema1.Table1 has been dropped at some point since the backup was created. When dbadmin performs the restore, Vertica re-creates Schema1.Table1. As the user performing the restore, dbadmin takes ownership of Schema1.Table1. Because Schema1 still exists, Alice retains ownership of the schema.

Scenario 2:

Schema1 is dropped, along with all contained objects. When dbadmin performs the restore, Vertica re-creates the schema and all contained objects. Dbadmin takes ownership of Schema1 and Schema1.Table1.

Scenario 3:

Schema1 and Schema1.Table1 both exist in the current database. When the user, dbadmin, rolls back to an earlier backup, the ownership of the objects remains unchanged. Alice owns Schema1 and Bob owns Schema1.Table1.

Scenario 4:

Schema1.Table1 exists and dbadmin wants to roll back to an earlier version. In the time since the backup was made, ownership of Schema1.Table1 has changed to Chris. When dbadmin restores Schema1.Table1, Alice remains owner of Schema1 and Chris remains owner of Schema1.Table1. The restore does not revert ownership of Schema1.Table1 from Chris to Bob.

Restoring Data When Node Type Is Not Permanent

The only node type on which Vertica supports data restoration is Permanent. You cannot restore data on the remaining node types:

- Ephemeral
- Execute
- Standby

To restore or replicate to these nodes, you must first change the [destination node type](#) to PERMANENT.

For more information, refer to [Setting Node Type](#).

Copying the Database to Another Cluster

You can use the `vbr` utility to copy the entire database to another Vertica cluster. This feature helps you perform tasks such as copying a database between a development and a production environment. Copying your database to another cluster is essentially a simultaneous backup and restore operation. The data is backed up from the source database cluster and restored to the destination cluster in a single operation.

Note: The `copycluster` task is not compatible with HDFS Storage Locations. `Copycluster` uses the Linux `rsync` command to copy files from the source cluster to the target cluster. HDFS storage backup and restore is based on use of snapshots. Data in an HDFS storage location is backed up to HDFS itself. Vertica cannot transfer data to a remote HDFS cluster the same way that it can for a Linux cluster.

The directory locations for the Vertica catalog, data, and temp directories must be identical on the source and target database. Use the following `vsql` query to view the source database directory locations. This example sets expanded display, for illustrative purposes, and lists the columns of most interest, `node_name`, `storage_path`, and `storage_usage`.

```
VMart=> \x  
Expanded display is on.
```

```
VMart=> select node_name,storage_path, storage_usage from disk_storage;
-[ RECORD 1 ]-+-----
node_name      | v_vmart_node0001
storage_path   | /home/dbadmin/VMart/v_vmart_node0001_catalog/Catalog
storage_usage  | CATALOG
-[ RECORD 2 ]-+-----
node_name      | v_vmart_node0001
storage_path   | /home/dbadmin/VMart/v_vmart_node0001_data
storage_usage  | DATA,TEMP
-[ RECORD 3 ]-+-----
node_name      | v_vmart_node0001
storage_path   | home/dbadmin/SSD/schemas
storage_usage  | DATA
-[ RECORD 4 ]-+-----
node_name      | v_vmart_node0001
storage_path   | /home/dbadmin/SSD/tables
storage_usage  | DATA
-[ RECORD 5 ]-+-----
node_name      | v_vmart_node0001
storage_path   | /home/dbadmin/SSD/schemas
storage_usage  | DATA
-[ RECORD 6 ]-+-----
node_name      | v_vmart_node0002
storage_path   | /home/dbadmin/VMart/v_vmart_node0002_catalog/Catalog
storage_usage  | CATALOG
-[ RECORD 7 ]-+-----
node_name      | v_vmart_node0002
storage_path   | /home/dbadmin/VMart/v_vmart_node0002_data
storage_usage  | DATA,TEMP
-[ RECORD 8 ]-+-----
node_name      | v_vmart_node0002
storage_path   | /home/dbadmin/SSD/tables
storage_usage  | DATA
.
.
.
```

Notice the directory paths for the Catalog, Data, and Tempstorage. These paths are the same on all nodes in the source database and must be the same in the target database.

Note: When copying a database to another cluster, if the target data is different, `vbr` overwrites all existing data. To retain existing data on the target cluster, create a full database backup of the target before invoking the `copycluster vbr` task.

Identifying Node Names for Target Cluster

Before you can configure the target cluster, you need to know the exact names that `adminTools` supplied to all nodes in the source database.

To see the node names, run a query such as:

```
VMart=> select node_name from nodes;
  node_name
-----
v_vmart_node0001
v_vmart_node0002
v_vmart_node0003
(3 rows)
```

You can also find the node names by running `Admintools` from the command line. For example, for the VMart database, you can enter a command such as:

```
$ /opt/vertica/bin/admintools -t node_map -d VMART
DATABASE | NODENAME          | HOSTNAME
-----
VMART    | v_vmart_node0001 | 192.168.223.xx
VMART    | v_vmart_node0002 | 192.168.223.yy
VMART    | v_vmart_node0003 | 192.168.223.zz
```

Configuring the Target Cluster

Configure the target to allow the source database to connect to it and restore the database. The target cluster must:

- Have the same number of nodes the source cluster.
- Have a database with the same name as the source database. The target database can be completely empty.
- Have the same node names as the source cluster. The nodes names listed in the `NODES` system tables on both clusters must match.
- Be accessible from the source cluster.
- Have the same database administrator account, and all nodes must allow a database administrator of the source cluster to login through SSH without a password.

Note: Passwordless access *within* the cluster is not the same as passwordless access *between* clusters. The SSH ID of the administrator account on the source cluster and the target cluster are likely not the same. You must configure each host in the target cluster to accept the SSH authentication of the source cluster.

- Have adequate disk space for the `vbr --task copycluster` command to complete.

Creating a Configuration File for CopyCluster

You must create a configuration file specifically for copying your database to another cluster. In the configuration file, specify the host names of nodes in the target cluster as the backup hosts. When you use the `copycluster` command, the `vbr` requires that you define the `backupHost`. However, the `vbr` ignores the `backupDir` option, and always stores the data in the catalog and data directories of the target database.

You cannot use an object-level backup with the `copycluster` command. Instead, you must perform a full database backup.

The following example shows how you can set up the `vbr` to copy a database on a 3-node cluster, `v_vmart`, to another cluster, `test-host`.

```
[Misc]
snapshotName = CopyVmart
restorePointLimit = 5
tempDir = /tmp/vbr
retryCount = 5
retryDelay = 1

[Database]
dbName = vmart
dbUser = dbadmin
dbPassword = password
dbPromptForPassword = False

[Transmission]
encrypt = False
checksum = False
port_rsync = 50000

[Mapping]
; backupDir is not used for cluster copy
v_vmart_node0001= test-host01
v_vmart_node0002= test-host02
v_vmart_node0003= test-host03
```

Copying the Database

You must stop the target cluster before you invoke `copycluster`.

To copy the cluster, run `vbr` from a node in the source database using the database administrator account. The following example demonstrates how you can copy a cluster using a configuration file located in the current directory.

```
$ vbr -t copycluster -c copycluster.ini
Starting copy of database VMART.
Participating nodes: vmart_node0001, vmart_node0002, vmart_node0003, vmart_node0004.
Enter vertica password:
Snapshotting database.
Snapshot complete.
Determining what data to copy.
[=====] 100%
Approximate bytes to copy: 987394852 of 987394852 total.
Syncing data to destination cluster.
[=====] 100%
Reinitializing destination catalog.
Copycluster complete!
```

If the copy cluster task is interrupted, the destination cluster retains any data files that have already transferred. If you attempt the operation again, Vertica does not need to resend these files.

Related Tasks

- [Configuring Backup Hosts](#)

Replicating Tables and Schemas to an Alternate Database

You can replicate Vertica tables and schemas from one database to alternate databases in your organization. Using this strategy helps you:

- Replicate objects to a secondary site.
- Copy tables and schemas between test, staging, and production clusters that have the same number of database nodes.

Important: The `vbr replicate` task does not support copying an entire database; use the `copycluster` task to replicate an entire database.

Advantages of Alternate Database Replication

Replicating objects is generally faster than exporting and importing objects. As with Vertica backup and restore, object replication is incremental. The first replication of an object

replicates the entire object. Subsequent replications copy only data that has changed since the last replication. Vertica replicates data as of the current epoch on the target database. Used with a cron job, you can replicate key objects to create a backup database.

In situations where the target database is down, or you plan to replicate the entire database, Micro Focus recommends that you try [Copying the Database to Another Cluster](#)

Catalog Size Limitations

Object level restores can fail if your catalog size is greater than five percent of the total memory available in the node performing the restore. In this situation, Micro Focus recommends restoring individual objects from the backup. For more information, refer to [Restoring Individual Objects from a Full or Object-Level Backup](#).

How DOWN Nodes Affect Replication

The effect of DOWN nodes on a replication task depends on whether they are present in the source or destination database.

Location	Effect on Replication
DOWN source nodes	Vertica can replicate objects from a source database containing DOWN nodes. If nodes in the source database are DOWN, set the corresponding nodes in the target database to DOWN as well.
DOWN destination nodes	Vertica can replicate objects when the destination database has DOWN nodes. If nodes in the destination database are DOWN, you must exclude the corresponding source database nodes using the <code>vbr --nodes</code> parameter.

Replication to Alternate Database Process Flow

To replicate objects to an alternate database, begin in the SOURCE database and complete the following process:

1. Verify Replication Requirements
2. Edit Your vbr Configuration File

3. Replicate Objects
4. Monitor Object Replication

Verify Replication Requirements

To replicate objects to an alternate database, verify that your source and target databases meet the following requirements.

Both the source and target databases must:

- Have the same Linux user associated with the dbadmin account on both the source and target databases.
- Be UP.
- Have the same number of nodes.
- Have the same exact version of Vertica.
- All nodes allowing a database administrator of the source cluster to login through SSH without a password.

Note: Passwordless access *within* the cluster is not the same as passwordless access *between* clusters. The SSH ID of the administrator account on the source cluster and the target cluster are likely not the same. You must configure each host in the target cluster to accept the SSH authentication of the source cluster.

Edit Your vbr Configuration File for Replication

The default configuration file that vbr creates does not contain all of the settings necessary to replicate objects to an alternate cluster.

Add the following parameters to the configuration file that you are using to replicate objects:

1. In the [\[Misc\] section](#) of your vbr configuration file, add the following parameter:

```
; Identify the objects that you want to replicate  
objects = schema.objectName
```

2. In the [\[Database\] section](#) of your vbr configuration file, add the following parameters:

```
; parameters used to replicate objects between databases
dest_dbName =
dest_dbUser =
dest_dbPromptForPassword =
```

If you are using a stored password, be sure to configure the `dest_dbPassword` parameter in your [Password Configuration File Reference](#).

3. In the [\[Mapping\] section](#), map your source nodes to your target hosts:

```
[Mapping]
v_source_node0001 = targethost01
v_source_node0002 = targethost02
v_source_node0003 = targethost03
```

Replicate Objects

To replicate objects, use the vbr task `replicate`. Specify this task in the following form:

```
vbr -t replicate -c configfile.ini
```

Monitor Object Replication

You can monitor object replication the following ways:

- Viewing vbr logs on the source database
- Checking database logs on the source and destination databases
- Querying [V_MONITOR.REMOTE_REPLICATION_STATUS](#) on the source database

Troubleshooting Backup and Restore

These tips can help you avoid issues related to backup and restore with Vertica and to troubleshoot any problems that occur.

Object Replication Fails

Confirm that you have excluded all DOWN nodes from your object replication.

If you do not exclude the DOWN node, replication fails with the following error:

```
Error connecting to a destination database node on the host <hostname>  
: <error> ...
```

Recovering the Database

Recovering a database can consist of any of the following:

- [Restarting Vertica on a host](#)
- [Restarting the Database](#)
- [Recovering the Cluster From a Backup](#)
- [Replacing Failed Disks](#)
- [Copying the Database to Another Cluster](#)
- [Exporting a Catalog](#) for support purposes
- [Recovering a node on a per-table basis](#)

You can [monitor a recovery](#) in progress by viewing log messages posted to the `vertica.log` file on each host.

See Also

- [Failure Recovery](#)

Failure Recovery

Vertica can restore the database to a fully functional state after one or more nodes in the system experiences a software- or hardware-related failure. Vertica recovers nodes by querying replicas of the data stored on other nodes. For example, a hardware failure can cause a node to lose database objects or to miss changes made to the database (INSERTs, UPDATEs, and so on) while offline. When the node comes back online, queries other nodes in the cluster to recover lost objects and catch up with database changes.

K-safety sets fault tolerance for the database cluster, where K can be set to 0, 1, or 2. The value of K specifies how many copies Vertica creates of [segmented projection](#) data. If K-safety for a database is set to 1 or 2, Vertica creates K+1 instances, or *buddies*, of each projection segment. Vertica distributes these buddies across the database cluster, such that projection data is protected in the event of node failure. If any node fails, the database can continue to process

queries so long as buddies of data on the failed node remain available elsewhere on the cluster.

Note: You can monitor the cluster state through the **View Database Cluster** state menu option.

Recovery Scenarios

Vertica begins the database recovery process when you restart failed nodes or the database. The mode of recovery for a K-safe database depends on the type of failure:

- One or more nodes in the database failed, but the database continued to operate.
- The database shut down cleanly.
- The database shut down uncleanly.

In the first two cases, node recovery is automatic; in the third case (unclean shutdown), recovery requires manual intervention by the database administrator. The following sections discuss these cases in greater detail.

Recovery of failed nodes

One or more nodes failed but the remaining nodes in the database filled in for them, so database operations continued without interruption. Use [Administration Tools](#) to restart failed nodes through the [Restart Vertica on Host](#) option. While restarted nodes recover their data from other nodes, their status is set to RECOVERING. Except for a short period at the end, the recovery phase has no effect on database transaction processing. After recovery is complete, the restarted nodes status changes to UP.

Recovery after clean shutdown

The database was shut down cleanly through Administration Tools. To restart the database, use the [Start Database](#) option. On restart, all nodes whose status was UP before the shutdown resume a status of UP. If the database contained one or more failed nodes on shutdown and they are now available, they begin the recovery process as described above.

Recovery after unclean shutdown

Reasons for unclean shutdown include:

- A critical node failed, leaving part of the database's data unavailable.
- A site-wide event such as a power failure causes all nodes to reboot.
- Vertica processes on the nodes exited due to a software or hardware failure.

Unclean shutdown can put the database in an inconsistent state—for example, Vertica might have been in the middle of writing data from WOS to disk at the time of failure, and this process was left incomplete. When you restart the database through the Administration Tools, Vertica determines that normal startup is not possible and uses the Last Good Epoch to determine when data was last consistent on all nodes. When you restart the database, Vertica prompts you to accept recovery with the suggested epoch. If accepted, the database recovers and all data changes after the Last Good Epoch are lost. If not accepted, startup is aborted.

Instead of accepting the recommended epoch, you can [recover from a backup](#). You can also choose an epoch that precedes the Last Good Epoch, through the Administration Tools Advanced Menu option Roll Back Database to Last Good Epoch. This is useful in special situations—for example the failure occurs during a batch of loads, where it is easier to restart the entire batch, even though some of the work must be repeated. In most cases, you should accept the recommended epoch.

Epochs and Node Recovery

The checkpoint epoch (CPE) for both the source and target projections are updated as ROS containers are moved. The start and end epochs of all storage containers, such as ROS containers, are modified to the commit epoch. When this occurs, the epochs of all columns without an actual data file rewrite advance the CPE to the commit epoch of `move_partitions_to_table`. If any nodes are down during the TM moveout and/or during the move partition, they will detect that there is storage to recover, and will recover from other nodes with the correct epoch upon rejoining the cluster.

Manual Recovery Notes

- You can manually recover a database where up to K nodes are offline—for example, they were physically removed for repair or not reachable at the time of recovery. When the missing nodes are restored, they recover and rejoin the cluster as described earlier in [Recovery Scenarios](#).
- You can manually recover a database if the nodes to be restarted can supply all partition segments, even if more than K nodes remain down at startup. In this case, all data is available from the remaining cluster nodes, so the database can successfully start.

- The default setting for the `HistoryRetentionTime` configuration parameter is 0, so Vertica only keeps historical data when nodes are down. This setting prevents use of the Administration Tools Roll Back Database to Last Good Epoch option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch that precedes the AHM. If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window to remove loaded data—for example:

```
=> ALTER DATABASE mydb SET HistoryRetentionTime = 86400;
```

See [Epoch Management Parameters](#) in the Administrator's Guide.

- When a node is down and manual recovery is required, it can take a full minute or longer for Vertica processes to time out while the system tries to form a cluster. Wait approximately one minute until the system returns the manual recovery prompt. Do not press CTRL-C during database startup.

See Also

[High Availability](#)

Restarting Vertica on a Host

When one node in a running database cluster fails, or if any files from the catalog or data directories are lost from any one of the nodes, you can check the status of failed nodes using either the Administration Tools or the Management Console.

Restarting Vertica on a Host Using the Administration Tools

1. Run Administration Tools.
2. From the Main Menu, select **Restart Vertica on Host** and click **OK**.
3. Select the database host you want to recover and click **OK**.

Note: You might see additional nodes in the list, which are used internally by the Administration Tools. You can safely ignore these nodes.

4. Verify recovery state by selecting **View Database Cluster State** from the **Main Menu**.

After the database is fully recovered, you can check the status at any time by selecting **View Database Cluster State** from the Administration Tools **Main Menu**.

Restarting Vertica on a Host Using the Management Console

1. Connect to a cluster node (or the host on which MC is installed).
2. Open a browser and [connect to MC](#) as an MC administrator.
3. On the MC **Home** page, double-click the running database under the **Recent Databases** section.
4. Within the **Overview** page, look at the node status under the Database sub-section and see if all nodes are up. The status will indicate how many nodes are up, critical, down, recovering, or other.
5. If a node is down, click **Manage** at the bottom of the page and inspect the graph. A failed node will appear in red.
6. Click the failed node to select it and in the Node List, click the **Start node** button.

Restarting the Database

If you lose the Vertica process on more than one node (for example, due to power loss), or if the servers are shut down without properly shutting down the Vertica database first, the database cluster indicates that it did not shut down gracefully the next time you start it.

The database automatically detects when the cluster was last in a consistent state and then shuts down, at which point an administrator can restart it.

From the Main Menu in the Administration Tools:


```
Database startup failed. Good epoch logs are available on all nodes.  
WARNING: if you say 'yes', changes made to database after  
'2008-01-28 16:49:31-05' (epoch 4203) will be permanently lost.  
  
Do you really want to restart the database from '2008-01-28 16:49:31-05' (epoch 4203)?  
  
< Yes > < No >
```

Caution: If you do not want to start from the last good epoch, you may instead restore the data from a backup and attempt to restart the database. For this to be useful, the backup must be more current than the last good epoch.

Vertica continues to initialize and recover all data prior to the last good epoch.

```
*** Restarting database QATESTDB at epoch 4203 ***  
Participating hosts:  
    rhel5-1  
    rhel5-2  
    rhel5-3  
    rhel5-4  
Checking vertica version on host rhel5-1  
Checking vertica version on host rhel5-2  
Checking vertica version on host rhel5-3  
Checking vertica version on host rhel5-4  
Processing host rhel5-1  
Processing host rhel5-2  
Processing host rhel5-3  
Processing host rhel5-4  
Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) site04: (RECOVERING)  
Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) site04: (RECOVERING)  
Node Status: site01: (RECOVERING) site02: (RECOVERING) site03: (RECOVERING) site04: (RECOVERING)  
Node Status: site01: (UP) site02: (UP) site03: (UP) site04: (UP)
```

If recovery takes more than a minute, you are prompted to answer <Yes> or <No> to "Do you want to continue waiting?"

When all the nodes' status have changed to RECOVERING or UP, selecting <No> lets you exit this screen and monitor progress via the Administration Tools Main Menu. Selecting <Yes> continues to display the database recovery window.

Note: Be sure to reload any data that was added after the last good epoch date to which you have recovered.

Recovering the Cluster From a Backup

To recover a cluster from a backup, refer to the following topics:

- [Backing Up and Restoring the Database](#)
- [Restoring Full Database Backups](#)

Phases of a Recovery

The phases of a Vertica recovery are the same regardless of whether you are recovering by table or node. In the case of a [recovery by table](#), tables become individually available as they complete the final phase. In the case of a recovery by node, the database objects only become available after the entire node completes recovery.

When you perform a recovery in Vertica, each recovered table goes through the following phases:

Order	Phase	Description	Lock Type
1	Historical	Vertica copies any historical data it may have missed while in a state of DOWN or INITIALIZING.	none
2	Historical Dirty	Vertica recovers any DML transactions that committed after the node or table began recovery.	none
3	Current Replay Delete	Vertica replays any delete transactions that took place during the recovery.	T-lock
4	Aggregate Projections	Vertica recovers any aggregate projections.	T-lock

After a table completes the last phase, Vertica considers it fully recovered. At this point, the table can participate in DDL and DML operations.

Monitoring Recovery

There are several ways to monitor database recovery:

- [Log files on each host](#)
- [Admintools \(View Database Cluster State\)](#)
- [System tables](#)
- [Cluster Status after recovery](#)

Exporting a Catalog

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the [EXPORT_CATALOG](#) function in the SQL Reference Manual for details.

Best Practices for Disaster Recovery

To protect your database from site failures caused by catastrophic disasters, maintain an off-site replica of your database to provide a standby. In case of disaster, you can switch database users over to the standby database. The amount of data loss between a disaster and fail over to the offsite replica depends on how frequently you save a full database backup.

The solution to employ for disaster recover depends upon two factors that you must determine for your application:

- **Recovery point objective (RPO):** How much data loss can your organization tolerate upon a disaster recovery?

- **Recovery time objective (RTO):** How quickly do you need to recover the database following a disaster?

Depending on your RPO and RTO, Vertica recommends choosing from the following solutions:

1. **Dual-load:** During each load process for the database, simultaneously load a second database. You can achieve this easily with off-the-shelf ETL software.
2. **Periodic Incremental Backups:** Use the procedure described in [Copying the Database to Another Cluster](#) to periodically copy the data to the target database. Remember that the script copies only files that have changed.
3. **Replication solutions provided by Storage Vendors:** Although some users have had success with SAN storage, the number of vendors and possible configurations prevent Micro Focus from providing support for SANs.

The following table summarizes the RPO, RTO, and the pros and cons of each approach:

	Dual Load	Periodic Incremental	Storage Replication
RPO	Up to the minute data	Up to the last backup	Recover to the minute
RTO	Available at all times	Available except when backup in progress	Available at all times
Pros	<ul style="list-style-type: none"> • Standby database can have different configuration • Can use the standby database for queries 	<ul style="list-style-type: none"> • Built-in scripts • High performance due to compressed file transfers 	Transparent to the database
Cons	<ul style="list-style-type: none"> • Possibly incur additional ETL licenses • Requires application logic to handle errors 	Need identical standby system	<ul style="list-style-type: none"> • More expensive • Media corruptions are also replicated

Query Performance with Failed Nodes

When a node fails within a Vertica cluster, the cluster detects the failure within milliseconds. The impact of a node failure on currently running queries depends on the [K-safety](#) level of your cluster and the location of your data within the cluster.

K-Safety 0 Cluster

If a node fails in a cluster with a K-safety of 0, any currently running queries roll back and the cluster shuts down. You cannot run further queries until you restore the failed node.

K-Safety 1 or Greater Cluster

The failure of a node can invalidate the query plan for a currently running query. If the running query does not require the failed node, the query continues to run. If a currently running query does require the failed node, the query rolls back. Vertica then automatically reconfigures the query plan to use alternate nodes and automatically retries the query. By default, Vertica makes three attempts to rerun the query before canceling it.

In clusters with a failed node, the database optimizer chooses different segmentations of buddy projections to distribute the workload throughout the cluster. Query performance within a cluster containing a failed node can, however, deteriorate, as one node must now do the work of two nodes. The extent of the performance impact varies depending on the queries that you run and the location of the data that you are querying. Query performance remains affected until the failed node has been completely restored.

Recovery By Table

Vertica supports node recovery on a per-table basis. Unlike a node-based recovery, recovering by table makes tables available as they recover, before the node itself is completely restored. You can [prioritize your most important tables](#) to ensure that they become available as soon as possible. Recovered tables support all DDL and DML operations.

To enhance recovery speed, Vertica recovers multiple tables in parallel. The maximum number of tables recoverable at one time is governed by the MAXCONCURRENCY parameter in the RECOVERY resource pool.

Once a node has fully recovered, it enables full Vertica functionality.

Recovery by table is enabled by default.

Note: Vertica does not support recovery by table for tables with pre-join projections.

Enabling Recovery by Table

With all the nodes in your cluster up, enable recovery by table with the following query:

```
SELECT SET_RECOVER_BY_TABLE('true');
```

Important: Do *not* disable recovery by table. SET_RECOVER_BY_TABLE has been deprecated and will be removed in a future release

Prioritizing Table Recovery

You can specify the order in which Vertica recovers tables. This feature ensures that your most critical tables become available as soon as possible. To specify the recovery order of your tables, assign an integer priority value. Tables with higher priority values recover first. For example, a table with a priority of 1000 is recovered before a table with a value of 500. Table priorities have the maximum value of a 64-bit integer.

If you do not assign a priority, or if multiple tables have the same priority, Vertica restores tables by [OID](#) order. Assign a priority with a query such as this:

```
=> SELECT set_table_recover_priority('avro_basic', '1000');
      set_table_recover_priority
-----
Table recovery priority has been set.
(1 row)
```

View assigned priorities with a query using this form:

```
SELECT table_name, recover_priority FROM v_catalog.tables;
```

The next example shows prioritized tables from the VMart sample database. In this case, the table with the highest recovery priorities are listed first (DESC). The shipping_dimension table has the highest priority and will be recovered first. (Example has hard Returns for display purposes.)

```
=> SELECT table_name AS Name, recover_priority from v_catalog.tables WHERE recover_priority > 1
      ORDER BY recover_priority DESC;
      Name          | recover_priority
-----+-----
shipping_dimension |          60000
```

```
warehouse_dimension |          50000  
employee_dimension  |          40000  
vendor_dimension    |          30000  
date_dimension      |          20000  
promotion_dimension |          10000  
iris2               |           9999  
product_dimension   |             10  
customer_dimension  |             10  
(9 rows)
```

Viewing Table Recovery Status

View general information about a recovery querying the `V_MONITOR.TABLE_RECOVERY_STATUS` table. You can also view detailed information about the status of the recovery the table being restored by querying the `V_MONITOR.TABLE_RECOVERIES` table.

Collecting Database Statistics

The Vertica cost-based query optimizer relies on data statistics to produce query plans. If statistics are incomplete or out-of-date, the optimizer is liable to use a sub-optimal plan to execute a query.

For example, you load timestamp data into a large table at regular intervals and query the table for the mostly recently loaded rows. The following scenarios influence how optimizer chooses a plan:

- You load days 1 through 15 into the table and run [ANALYZE_STATISTICS](#). When you next run a query that requests yesterday's data by filtering on the timestamp column, the optimizer chooses an optimized query plan.
- The next day, you load day 16 data and run the same query. If you do not run [ANALYZE_STATISTICS](#) again, the optimizer might conclude that the predicate results in only one row being returned because the date range falls outside the histogram range and the data becomes stale. When the optimizer detects that statistics are not current for a particular predicate—in this case, because a timestamp predicate is outside a histogram's boundary)—Vertica uses other considerations to plan the query, such as FK-PK constraints.

In this case, you can run [ANALYZE_STATISTICS](#) after loading new data on day 16. You can also look for statistics in the [EXPLAIN](#)-generated query plan. For example, when statistics are outside a histogram's boundaries, the query plan is annotated with a status.

Vertica provides a number of tools like [ANALYZE_STATISTICS](#) for analyzing and updating database statistics. For an overview, see [Database Statistics Tools](#).

Database Statistics Tools

Statistics collection is a cluster-wide operation that accesses data using a historical query (at epoch latest) without any locks. After statistics are computed, they are stored in the catalog and replicated on all nodes. The storage operation requires a brief, exclusive lock on the catalog, similar to when a DDL operation occurs. In fact, these operations require a COMMIT for the current transaction.

Vertica provides a number of tools for analyzing and updating database statistics:

Tool	Description
ANALYZE_	Collects a statistical data sampling.

Tool	Description
STATISTICS	
ANALYZE_ROW_COUNT	Invoked through the Vertica function DO_TM_TASK , collects projection row counts.
ANALYZE_EXTERNAL_ROW_COUNT	Collects row count data for external tables.
EXPORT_STATISTICS	Generates database statistics in XML format from data previously collected by ANALYZE_STATISTICS .
PROJECTION_COLUMNS	Monitors information about projection columns, such as encoding type, sort order, type of statistics, and the time at which columns statistics were last updated.

For more information about these analysis tools, see [Analyzing Row Counts](#) and [Getting Statistics](#).

See also

For information about tools to monitor and analyze query performance, see [Analyzing Workloads](#).

For information about managing statistics, see descriptions of the following Vertica functions in the SQL Reference Manual:

- [EXPORT_STATISTICS](#)
- [IMPORT_STATISTICS](#)
- [DROP_STATISTICS](#)
- [DROP_EXTERNAL_ROW_COUNT](#)

Analyzing Row Counts

Vertica lets you obtain row counts for projections and for external tables, through [ANALYZE_ROW_COUNT](#) and [ANALYZE_EXTERNAL_ROW_COUNT](#), respectively.

Projection Row Count

`ANALYZE_ROW_COUNT` is a lightweight operation that automatically collects the number of rows in a projection every 60 seconds to collect a minimal set of statistics and aggregate row counts calculated during loads. You can invoke this function through the function `DO_TM_TASK`. For example:

```
=> SELECT DO_TM_TASK ('analyze_row_count', 'public.Emp_Dimension_b0');
```

To change the collection interval, set the configuration parameter `AnalyzeRowCountInterval` (see [Configuration Parameters](#)). For example, you can change the collection interval to 1 hour (3600 seconds) as follows:

```
=> ALTER DATABASE mydb SET AnalyzeRowCountInterval = 3600;
```

External Table Row Count

`ANALYZE_EXTERNAL_ROW_COUNT` calculates the exact number of rows in an external table. The optimizer uses this count to optimize for queries that access external tables. This is especially useful when an external table participates in a join. This function enables the optimizer to identify the smaller table to use as the inner input to the join, and facilitate better query performance.

The following query calculates the exact number of rows in the external table `loader_rejects`:

```
=> SELECT ANALYZE_EXTERNAL_ROW_COUNT('loader_rejects');
ANALYZE_EXTERNAL_ROW_COUNT
-----
0
```

Getting Statistics

`ANALYZE_STATISTICS` collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table. You call this function as follows:

```
ANALYZE_STATISTICS ('[schema.]table'
                    [, 'column[,...]']
                    [, percent ] )
```

If you call this function with a single empty string argument, Vertica collects statistics for all database tables and their projections:

```
=> SELECT ANALYZE_STATISTICS ('');
```

Operations

ANALYZE_STATISTICS performs the following operations:

- Performs fast data sampling, which expedites analysis of relatively small tables with a large number of columns.
- Includes data from WOS.
- Recognizes deleted data instead of ignoring delete markers.
- Records in system table [PROJECTION_COLUMNS](#) the last time statistics were run for a table.
- Enables table analysis tables on a per-column basis for improved performance.

Sampling Size

ANALYZE_STATISTICS constructs a column histogram from a set of rows that it randomly selects from all collected data. Regardless of the percentage setting, the function always creates a statistical sample that contains up to (approximately) the smaller of:

- 2^{17} (131,072) rows
- Number of rows that fit in 1 GB of memory

If a column has fewer rows than the maximum sample size, ANALYZE_STATISTICS reads all rows from disk and analyzes the entire column.

Note: The data collected in a sample range does not indicate how data should be distributed.

The following table shows how ANALYZE_STATISTICS, when set to different percentages, obtains a statistical sample from a given column:

Number of column rows	%	Number of rows read	Number of sampled rows
$\leq \text{max-sample-size}$	20	All	All

Number of column rows	%	Number of rows read	Number of sampled rows
400K	10	<i>max-sample-size</i>	<i>max-sample-size</i>
4000K	10	400K	<i>max-sample-size</i>

Note: When a column specified for `ANALYZE_STATISTICS` is first in a projection's sort order, the function reads all data from disk to avoid a biased sample.

Data Collection Percentage

If `ANALYZE_STATISTICS` omits specifying a percentage, Vertica collects a fixed 10-percent sample of statistical data from disk. Specifying a percentage of data to read from disk gives you more control over deciding between sample accuracy and speed.

The percentage of data you collect affects collection time and accuracy:

- A smaller percentage is faster but returns a smaller data sample, which might compromise histogram accuracy.
- A larger percentage reads more data off disk. Data collection is slower, but a larger data sample enables greater histogram accuracy.

Evaluating Results

After you collect the desired statistics, run the Workload Analyzer to retrieve hints about under-performing queries and their root causes, and obtain tuning recommendations. For more information, see [Analyzing Workloads](#).

Examples

Compute statistics on all projections in the `VMart` database:

```
=> SELECT ANALYZE_STATISTICS ('');
ANALYZE_STATISTICS
-----
0
(1 row)
```

Compute statistics on a single table:

```
=> SELECT ANALYZE_STATISTICS ('shipping_dimension');
ANALYZE_STATISTICS
-----
                0
(1 row)
```

Compute statistics on a single column across all projections for a table:

```
=> SELECT ANALYZE_STATISTICS('shipping_dimension','shipping_key');
ANALYZE_STATISTICS
-----
                0
(1 row)
```

Collect data from the entire disk by setting the percent parameter to 100:

```
=> SELECT ANALYZE_STATISTICS ('shipping_dimension', 'shipping_key', 100);
ANALYZE_STATISTICS
-----
                0
(1 row)
```

Collect data on all projections for `shipping_dimension` from 20 percent of the disk:

```
=> SELECT ANALYZE_STATISTICS ('shipping_dimension', 20);
ANALYZE_STATISTICS
-----
                0
(1 row)
```

Reporting Statistics

Vertica supplies hints about statistics two ways:

- The [EXPLAIN](#)-generated query plan is annotated with a status. See [Reacting to Stale Statistics](#).
- The last time [ANALYZE_STATISTICS](#) was run for a table is recorded, so that subsequent calls to the function are optimized. This is useful during the database design process because if Database Designer does not collect statistics when adding design tables, it generates a warning indicating that statistics are old. You can then decide whether to run [ANALYZE_STATISTICS](#) before you proceed with the design.

Two columns in the [V_CATALOG.PROJECTION_COLUMNS](#) system table capture statistical information, as follows:

- `STATISTICS_TYPE` returns the type of statistics the column contains, one of the following: `NONE`, `ROWCOUNT`, or `FULL`.
- `STATISTICS_UPDATED_TIMESTAMP` returns the last time statistics were collected in this table.

Determining When Statistics Were Last Updated

System table `PROJECTION_COLUMNS` returns statistics about projection columns, including the type of statistics and when they were last updated.

For example, the following sample schema defines a table named `trades`, which groups the highly-correlated columns `bid` and `ask` and stores the `stock` column separately:

```
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);
=> CREATE PROJECTION trades_p (
    stock ENCODING RLE, GROUPED(bid ENCODING DELTAVAL, ask))
    AS (SELECT * FROM trades) ORDER BY stock, bid;
=> INSERT INTO trades VALUES('acme', 10, 20);
=> COMMIT;
```

Query the `PROJECTION_COLUMNS` table for table `trades`:

```
=> \x
Expanded display is on.
=> SELECT * FROM PROJECTION_COLUMNS WHERE table_name = 'trades';
```

The `statistics_type` column returns `NONE` for all three columns in the `trades` table. The `statistics_updated_timestamp` field is empty because statistics have not yet been run on this table.

```
-[ RECORD 1 ]-----+-----
projection_id      | 45035996273718838
projection_name    | trades_p
projection_column_name | stock
column_position   | 0
sort_position     | 0
column_id         | 45035996273718840
data_type         | char(5)
encoding_type     | RLE
access_rank      | 0
group_id         | 0
table_schema     | public
table_id         | 45035996273718836
table_name       | trades
table_column_id  | 45035996273718836-1
```

```

table_column_name      | stock
statistics_type        | NONE
statistics_updated_timestamp |
is_expression          | f
is_aggregate           | f
partition_by_position  |
order_by_type          |
column_expression      |
-[ RECORD 2 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | bid
column_position        | 1
sort_position         | 1
column_id              | 45035996273718842
data_type              | int
encoding_type         | DELTAVAL
access_rank           | 0
group_id              | 45035996273718844
table_schema          | public
table_id              | 45035996273718836
table_name            | trades
table_column_id       | 45035996273718836-2
table_column_name     | bid
statistics_type        | NONE
statistics_updated_timestamp |
is_expression          | f
is_aggregate           | f
partition_by_position  |
order_by_type          |
column_expression      |
-[ RECORD 3 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | ask
column_position        | 2
sort_position         |
column_id              | 45035996273718846
data_type              | int
encoding_type         | AUTO
access_rank           | 0
group_id              | 45035996273718844
table_schema          | public
table_id              | 45035996273718836
table_name            | trades
table_column_id       | 45035996273718836-3
table_column_name     | ask
statistics_type        | NONE
statistics_updated_timestamp |

```

Now, run statistics on the stock column:

```
=> SELECT ANALYZE_STATISTICS('trades.stock');
```

The system returns 0 for success:

```
-[ RECORD 1 ]-----+--
ANALYZE_STATISTICS | 0
```

Now query PROJECTION_COLUMNS again:

```
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
```

This time, `statistics_type` changes to FULL for the `trades.stock` column (representing full statistics were run). The `statistics_updated_timestamp` column returns the time the stock columns statistics were updated. The timestamp for the `bid` and `ask` columns have not changed because statistics were not run on those columns. Also, the `bid` and `ask` columns changed from NONE to ROWCOUNT because Vertica automatically updates ROWCOUNT statistics from time to time. The statistics are created by looking at existing catalog metadata.

```
-[ RECORD 1 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | stock
column_position       | 0
sort_position         | 0
column_id             | 45035996273718840
data_type             | char(5)
encoding_type        | RLE
access_rank          | 0
group_id             | 0
table_schema         | public
table_id             | 45035996273718836
table_name           | trades
table_column_id      | 45035996273718836-1
table_column_name    | stock
statistics_type       | FULL
statistics_updated_timestamp | 2012-12-08 13:52:04.178294-05
is_expression        | f
is_aggregate         | f
partition_by_position |
order_by_type        |
column_expression    |
-[ RECORD 2 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | bid
column_position       | 1
sort_position         | 1
column_id             | 45035996273718842
data_type             | int
encoding_type        | DELTAVAL
access_rank          | 0
group_id             | 45035996273718844
table_schema         | public
table_id             | 45035996273718836
table_name           | trades
table_column_id      | 45035996273718836-2
table_column_name    | bid
statistics_type       | ROWCOUNT
statistics_updated_timestamp | 2012-12-08 13:51:20.016465-05
```

```

is_expression          | f
is_aggregate           | f
partition_by_position  |
order_by_type          |
column_expression      |
-[ RECORD 3 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | ask
column_position        | 2
sort_position         |
column_id              | 45035996273718846
data_type              | int
encoding_type          | AUTO
access_rank            | 0
group_id              | 45035996273718844
table_schema           | public
table_id               | 45035996273718836
table_name             | trades
table_column_id        | 45035996273718836-3
table_column_name      | ask
statistics_type        | ROWCOUNT
statistics_updated_timestamp | 2012-12-08 13:51:20.016475-05
is_expression          | f
is_aggregate           | f
partition_by_position  |
order_by_type          |
column_expression      |

```

If you run statistics on the bid column and then query this system table again, only RECORD 2 is updated:

```

=> SELECT ANALYZE_STATISTICS('trades.bid');
-[ RECORD 1 ]-----+---
ANALYZE_STATISTICS | 0
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
-[ RECORD 1 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | stock
column_position        | 0
sort_position         | 0
column_id              | 45035996273718840
data_type              | char(5)
encoding_type          | RLE
access_rank            | 0
group_id              | 0
table_schema           | public
table_id               | 45035996273718836
table_name             | trades
table_column_id        | 45035996273718836-1
table_column_name      | stock
statistics_type        | FULL
statistics_updated_timestamp | 2012-12-08 13:52:04.178294-05
is_expression          | f
is_aggregate           | f
partition_by_position  |
order_by_type          |

```

```

column_expression      |
-[ RECORD 2 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | bid
column_position       | 1
sort_position         | 1
column_id             | 45035996273718842
data_type             | int
encoding_type        | DELTAVAL
access_rank          | 0
group_id             | 45035996273718844
table_schema         | public
table_id             | 45035996273718836
table_name           | trades
table_column_id      | 45035996273718836-2
table_column_name    | bid
statistics_type      | FULL
statistics_updated_timestamp | 2012-12-08 13:53:23.438447-05
is_expression        | f
is_aggregate         | f
partition_by_position |
order_by_type        |
column_expression    |
is_expression        | f
is_aggregate         | f
partition_by_position |
order_by_type        |
column_expression    |
-[ RECORD 3 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | ask
column_position       | 2
sort_position         |
column_id             | 45035996273718846
data_type             | int
encoding_type        | AUTO
access_rank          | 0
group_id             | 45035996273718844
table_schema         | public
table_id             | 45035996273718836
table_name           | trades
table_column_id      | 45035996273718836-3
table_column_name    | ask
statistics_type      | ROWCOUNT
statistics_updated_timestamp | 2012-12-08 13:51:20.016475-05
is_expression        | f
is_aggregate         | f
partition_by_position |
order_by_type        |
column_expression    |

```

You can query the `statistics_updated_timestamp` column to see when columns were updated:

```

=> \x
Expanded display is off.

```

```
=> SELECT ANALYZE_STATISTICS('trades');
ANALYZE_STATISTICS
-----
                0
(1 row)
=> SELECT projection_column_name, statistics_type,
       statistics_updated_timestamp
       FROM PROJECTION_COLUMNS where table_name = 'trades';
projection_column_name | statistics_type | statistics_updated_timestamp
-----+-----+-----
stock                  | FULL           | 2012-12-08 13:54:27.428622-05
bid                    | FULL           | 2012-12-08 13:54:27.428632-05
ask                    | FULL           | 2012-12-08 13:54:27.428639-05
(3 rows)
```

Reacting to Stale Statistics

During predicate selectivity estimation, the query optimizer can identify when histograms are not available or are out of date. If the value in the predicate is outside the histogram's maximum range, the statistics are stale. If no histograms are available, then no statistics are available to the plan.

When the optimizer detects stale or no statistics, such as when it encounters a column predicate for which it has no histogram, the optimizer performs the following actions:

- Displays and logs a message that you should run [ANALYZE_STATISTICS](#).
- Annotates [EXPLAIN](#)-generated query plans with a statistics entry.
- Ignores stale statistics when it generates a query plan. The optimizer uses other considerations to create a query plan, such as FK-PK constraints.

For example, the following query plan fragment shows no statistics (histograms unavailable):

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
```

The following query plan fragment shows that the predicate falls outside the histogram range:

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 35, Rows: 1 (PREDICATE VALUE OUT-OF-RANGE)]
```

You can get information about which table column has no statistics by querying a system table; for example, view the timestamp for when statistics were last run by querying system table [PROJECTION_COLUMNS](#).

Example

1. Run full statistics on table trades:

```
=> SELECT ANALYZE_STATISTICS('trades'); ANALYZE_STATISTICS
-----
                                0
(1 row)
```

2. Query columns `column_name`, `statistics_type`, and `statistics_updated_timestamp` from projection `PROJECTION_COLUMNS`.

```
=> SELECT projection_column_name, statistics_type, statistics_updated_timestamp
FROM PROJECTION_COLUMNS where table_name = 'trades';
projection_column_name | statistics_type | STATISTICS_UPDATED_TIMESTAMP
-----+-----+-----
stock                  | FULL           | 2011-03-31 13:39:16.968177-04
bid                    | FULL           | 2011-03-31 13:39:16.96885-04
ask                    | FULL           | 2011-03-31 13:39:16.968883-04
(3 rows)
```

You can also query the `PROJECTIONS` column `HAS_STATISTICS`, which returns true only when all non-epoch columns for a table have full statistics. Otherwise, the column returns false.

See Also

[Analyzing Workloads](#)

Canceling Statistics Collection

To cancel statistics collection mid analysis, execute CTRL-C on vsql or call the `INTERRUPT_STATEMENT()` function.

If you want to remove statistics for the specified table or type, call the `DROP_STATISTICS()` function.

Caution: After you drop statistics, it can be time consuming to regenerate them.

Best Practices for Statistics Collection

When you query a table, the query optimizer requires representative statistics on its data in order to choose the best query plan. For many applications, statistics do not need to be accurate to the latest minute. Vertica regularly calls `ANALYZE ROW COUNT`, which collects partial statistics and supplies enough data to satisfy many optimizer requirements.

Requesting Statistics

You can explicitly request Vertica to gather detailed information on the data of individual tables and their columns, by calling `ANALYZE_STATISTICS`. This function analyzes distribution of column data and storage usage across all projections. Without this data, the query optimizer assumes uniform distribution of data values and equal storage usage for all projections, and creates query plans accordingly.

You can specify the size of the data sample—by default, 10 percent. You can also narrow the scope of the collection by specifying individual columns.

You should call `ANALYZE_STATISTICS` on a table or individual columns when one or more of following conditions are true:

- Data is bulk loaded for the first time.
- A new projection is refreshed.
- The number of rows changes significantly.
- A new column is added to the table.
- Column minimum/maximum values change significantly.
- New primary key values with referential integrity constraints are added . The primary key and foreign key tables should be re-analyzed.
- Table size notably changes relative to other tables it is joined to—for example, a table that was 50 times larger than another table is now only five times larger.
- A notable deviation in data distribution necessitates recalculating histograms—for example, an event causes abnormally high levels of trading for a particular stock.
- The database is inactive for an extended period of time.

Overhead Considerations

Running `ANALYZE_STATISTICS` is an efficient but potentially long-running operation. You can run it concurrently with queries and loads in a production environment. However, the function can incur considerable overhead on system resources (CPU and memory), at the expense of queries and load operations. To minimize overhead, run `ANALYZE_STATISTICS` on individual tables rather than the entire database, or on individual table columns.

Related Tools

You can diagnose and resolve many statistics-related issues by calling [ANALYZE_WORKLOAD](#), which returns tuning recommendations.

If you update statistics and find that the query still performs poorly, run your query through the Database Designer and choose incremental as the design type. See [Incremental Design](#).

Using Diagnostic Tools

Micro Focus provides several diagnostic tools. This section includes the following.

- [Determining Your Version of Vertica](#)
- [Collecting Diagnostics \(scrutinize Command\)](#)
- [Exporting a Catalog](#)
- [Exporting Profiling Data](#)

Determining Your Version of Vertica

To determine which version of Vertica is installed on a host, log in to that host and type:

```
$ rpm -qa | grep vertica
```

The command returns the name of the installed package, which contains the version and build numbers. The following example indicates that both Vertica 7.0.x and Management Console 7.0.x are running on the targeted host:

```
[dbadmin@myhost01 ~]$ rpm -qa | grep vertica  
vertica-7.0.0-0  
vertica-console-7.0.0-0.x86_64
```

When you are logged in to your Vertica Analytics Platform database, you can also run a query for the version only, by running the following command:

```
dbadmin=> SELECT version();  
          version  
-----  
Vertica Analytic Database v7.0.0-0  
(1 row)
```

Collecting Diagnostics (scrutinize Command)

The diagnostics tool `scrutinize` collects a broad range of information from a Vertica cluster. It also supports a range of options that let you control the amount and type of data that is collected. Collected data can include but is not limited to:

- Host diagnostics and configuration data
- Run-time state (number of nodes up or down)
- Log files from the installation process, the database, and the administration tools (such as, `vertica.log`, `dbLog`, `/opt/vertica/log/adminTools.log`)
- Error messages
- Database design
- System table information, such as system, resources, workload, and performance
- Catalog metadata, such as system configuration parameters
- Backup information

Requirements

`scrutinize` requires that a cluster be configured to support the Administration Tools utility. If Administration Tools cannot run on the initiating host, then `scrutinize` cannot run on that host.

Further Information

- [Running scrutinize](#)
- [Limiting the Scope of scrutinize Data Collection](#)
- [Troubleshooting scrutinize](#)
- [Uploading scrutinize Results to Support](#)

Running scrutinize

You can run `scrutinize` unqualified by any options:

```
$ /opt/vertica/bin/scrutinize
```

When run without options, `scrutinize` collects a wide range of information from all cluster nodes. It stores the results in a `.tar` file (`VerticaScrutinizeNumericID.tar`), with

minimal effect on database performance. `scrutinize` output can help diagnose most issues and yet reduces upload size by omitting fine-grained profiling data.

`scrutinize` requires temporary disk space where it can collect data before posting the final compressed (`.tar`) output. How much space depends on variables such as the size of the Vertica log and extracted system tables, as well as user-specified options that limit the scope of information collected. Before `scrutinize` runs, it verifies that the temporary directory contains at least 1 GB of space; however, the amount that it actually needs can be much higher. By default, `scrutinize` uses `/opt/vertica/tmp` as the temporary directory. To avoid problems, you can create a temporary directory with as much space as you think `scrutinize` is likely to need. Then, specify that `scrutinize` use this temporary directory through its `--tmpdir` option.

You can run `scrutinize` with one or more options that limit the scope of information that it collects. For example, you can limit the time span of collected data, or collect data only from specific cluster nodes. These options can help you narrow the focus of a collection to information that pertains to a specific issue; they can also reduce output size and improve performance. For more information on these options, see [Limiting the Scope of scrutinize Data Collection](#).

Privileges

You must have database administrator (`dbadmin`) privileges to run `scrutinize`. If you run `scrutinize` as `root` when the `dbadmin` user exists, Vertica returns the following error:

```
[root@host01 ~]# /opt/vertica/bin/scrutinize
Root user is not allowed to use this tool.
Try again as the DB administrative user.
```

Command Options

Option	Description
<code>--auth-upload=<i>url</i></code> <code>-A <i>url</i></code> <code>--url=<i>url</i></code> <code>-u <i>url</i></code>	<p>The <code>--auth-upload</code> and <code>--url</code> options both post <code>scrutinize</code> output to a Vertica support-provided URL, where <i>url</i> specifies the URL or FTP address supplied by Vertica Customer Support for file upload:</p> <ul style="list-style-type: none">• <code>--auth-upload</code> uses your Vertica license to authenticate with the Vertica server.

Option	Description
	<ul style="list-style-type: none"> • <code>--url</code> requires <i>url</i> to include a user name and password that is supplied by Vertica Customer Support. <p>For more information, see Uploading scrutinize Results to Support.</p>
<p><code>--begin=timestamp</code> <code>--end=timestamp</code></p>	<p>Used together or separately, the <code>--begin</code> and <code>--end</code> options narrow the time frame of diagnostic data that is collected from <code>vertica.log</code> and <code>editor.log</code>:</p> <ul style="list-style-type: none"> • <code>--begin</code> specifies to collect log data from the specified begin time to the specified end time or log end. • <code>--end</code> specifies to collect data from the specified begin time or log start to the specified end time. <p>Note: These options cannot be combined with the <code>--include_gzlogs</code> option.</p> <p>You can specify <i>timestamp</i> arguments as offsets from the current time or as absolute times:</p> <ul style="list-style-type: none"> • Offset time format: '<i>DdHhMmSs</i>' For example: '3d2h1m' • Absolute time format: '<i>YYYY-MM-DD HH:MM:SS</i>' For example: '2014-01-02 13:29:59' <p>Examples</p> <pre>\$ scrutinize --begin='2014-01-01 00:00:00' --end='2014-01-07 23:59:59' \$ scrutinize --begin='2014-01-01 00:00:00' \$ scrutinize --end='2014-01-07 23:59:59' \$ scrutinize --begin='2014-09-23 00:00:00' --end='1d'</pre>
<p><code>--by-second</code> <code>--by-minute=boolean-value</code></p>	<p>Specifies the granularity of information that is collected from Data Collector tables with one of the following options:</p> <ul style="list-style-type: none"> • <code>--by-second</code>: Highest level of granularity, specifies to collect data down to the second.

Option	Description
	<ul style="list-style-type: none"> • <code>--by-minute=<i>boolean-value</i></code> where <i>boolean-value</i> is set to one of the following: <ul style="list-style-type: none"> ■ <code>{yes on}</code>: The default setting, specifies to collect data down to the minute. ■ <code>{no off}</code>: The lowest level of granularity, specifies to collect data down to the hour. <p>For example, the following command collects the last five days of data down to the hour:</p> <pre style="background-color: #f0f0f0; padding: 5px;">\$ scrutinize --begin=5d --by-minute=no</pre> <p>This command collects the last hour's data down to the second:</p> <pre style="background-color: #f0f0f0; padding: 5px;">\$ scrutinize --begin=1h --by-second</pre>
<pre>--database=DB -d DB</pre>	<p>Gathers diagnostics for the specified database.</p> <p>This option is required only if the following conditions are true:</p> <ul style="list-style-type: none"> • Multiple databases are defined on the cluster. • More than one database is active, or no database is active. <p>If you omit this option when these conditions are true, <code>scrutinize</code> returns a message that asks you to specify a database.</p> <p>Example</p> <pre style="background-color: #f0f0f0; padding: 5px;">\$ scrutinize -d VMart</pre>
<pre>--debug</pre>	<p>Increases the amount of information written to the log.</p>
<pre>--diag-dump</pre>	<p>Limits the collection to database design, system tables, and Data Collector tables. Use this option to collect data to analyze system performance.</p>
<pre>--diagnostics</pre>	<p>Limits the collection to log file data and output from commands that are run against Vertica and its host system. Use this option to collect data to evaluate unexpected behavior in your Vertica system.</p>

Option	Description
<pre>--exclude-tasks=<i>tasks</i> -X <i>tasks</i></pre>	<p>Excludes one or more of the specified types of tasks from the diagnostics collection, where <i>tasks</i> is a comma-separated list of the task types to exclude. To exclude all default tasks, supply the argument <code>all</code>.</p> <p>Important: Use <code>--exclude-tasks</code> only in consultation with your Vertica Customer Support contact.</p> <p>For detailed information, see Limiting the Scope of scrutinize Data Collection.</p>
<pre>--get-files <i>file-list</i></pre>	<p>Specifies extra files to collect, including globs, where <i>file-list</i> is a semicolon-delimited list of files.</p>
<pre>--help -h</pre>	<p>Outputs all <code>scrutinize</code> options to the console.</p>
<pre>--hosts=<i>host-list</i> -n <i>host-list</i></pre>	<p>Collects diagnostics for the specified hosts, where <i>host-list</i> is a comma-separated list of IP addresses or host names. By default, <code>scrutinize</code> collects diagnostics on all cluster hosts.</p> <p>To collect data only from the node on which <code>scrutinize</code> was invoked, use the <code>--local_diags (-s)</code> option.</p> <p>Example</p> <pre>scrutinize -n host1,host2,host3 scrutinize --hosts=127.0.0.1,host_3,host_1</pre>
<pre>--include_gzlogs=<i>num-files</i> -z <i>num-files</i></pre>	<p>Specifies to include <i>num-files</i> rotated log files (<code>vertica.log*.gz</code>) in the <code>scrutinize</code> output, where <i>num-files</i> can be one of the following:</p> <ul style="list-style-type: none"> • An integer specifies how many of the most recent rotated log files to collect. • <code>all</code> specifies to collect all rotated log files. <p>By default, <code>scrutinize</code> includes one rotated log file.</p> <p>Note: This option cannot be combined with the <code>--begin</code> and <code>--end</code> options.</p> <p>Example</p>

Option	Description
	<p><code>scrutinize</code> includes the four most recent rotated log files:</p> <pre data-bbox="553 338 1403 401">\$ scrutinize --include_gzlogs 4</pre>
<p><code>--include-ros-info</code></p>	<p>Includes ROS related information from system tables.</p>
<p><code>--local_diags</code> <code>-s</code></p>	<p>Gathers diagnostics from the host where <code>scrutinize</code> was invoked. By default, <code>scrutinize</code> collects data from all nodes in the cluster.</p> <p>Tip: To collect data from multiple nodes in the cluster, use the <code>--hosts (-n)</code> option.</p>
<p><code>--log-limit <i>Limit</i></code></p>	<p>Limits how much data is collected from Vertica logs, where <i>Limit</i> specifies, in gigabytes, how much log data to collect, starting from the most recent log entry. By default, 1 GB of log data is collected.</p>
<p><code>--message= <i>message</i></code> <code>-m <i>message</i></code></p>	<p>Includes a message in the <code>scrutinize</code> output, in the file <code>reason.txt</code>. Messages are typically included when you upload <code>scrutinize</code> to Vertica Customer Support.</p> <p>You can set a message several ways:</p> <ul style="list-style-type: none"> • Supply a message string enclosed by double quotation marks ("). • Supply the path to a file enclosed by double quotation marks. <code>scrutinize</code> includes the file contents in its output file. • Use the PROMPT keyword to open an input stream. <code>scrutinize</code> reads input until you type a period (.) on a new line. This closes the input stream, and <code>scrutinize</code> writes the message to the collected output. <p>For detailed information and examples, see Uploading scrutinize Results to Support.</p>
<p><code>--no-active-queries</code> <code>--with-active-queries</code></p>	<p>Specifies whether to collect diagnostic information from system tables and Data Collector tables about currently running queries. By default, <code>scrutinize</code> always collects active query information</p>

Option	Description
	<p>(<code>--with-active-queries</code>).</p>
<p><code>--no-containers</code></p>	<p>Omits running system table queries from the <code>scrutinize</code> collection. Omitting these queries can help reduce <code>scrutinize</code> run time.</p>
<p><code>--no-package-info</code></p>	<p>Excludes information collected about Vertica add-on packages like Vertica Kafka.</p>
<p><code>--output_dir=path</code> <code>-o path</code></p>	<p>Redirects output to a location that is not the current directory. By default, <code>scrutinize</code> places its output in the current directory.</p> <p>Example</p> <pre data-bbox="553 768 1401 831">\$ scrutinize --output_dir="/my_diagnostics/"</pre>
<p><code>--password='password'</code> <code>-P 'password'</code></p>	<p>Specifies the database password. <code>scrutinize</code> supports <code>VSQL_PASSWORD</code>.</p> <p>Omitting this argument on a password-protected database returns a warning. Enclose the password with single quotation marks (<code>'</code>) to protect passwords with special characters.</p> <p>Use this option if the Vertica administrator account (default <code>dbadmin</code>) has password authentication.</p> <p>Example</p> <pre data-bbox="553 1262 1401 1373">\$ scrutinize -P '@password**' \$ scrutinize --password='\$password1*'</pre>
<p><code>--tasks=tasks</code> <code>-T tasks</code></p>	<p>Specifies that <code>scrutinize</code> gather diagnostics on one or more specified tasks, as specified in a file or JSON list.</p> <p>Note: Use this option only in consultation with Vertica Customer Support.</p>
<p><code>--type=type</code> <code>-t type</code></p>	<p>Specifies the type of diagnostics collection to perform, where <code>type</code> can be set to one of the following arguments:</p> <ul data-bbox="553 1759 1122 1835" style="list-style-type: none"> • <code>profiling</code>: Gather profiling data. • <code>context</code>: Gather summary information.

Option	Description
<code>--tmpdir=path</code>	<p>Specifies the temporary directory to use on all nodes. By default, the temporary directory is in the Administration Tools directory <code>/tmp</code>. Use this option to ensure that the temporary directory has enough free space so <code>scrutinize</code> can complete execution.</p> <p>The following requirements apply:</p> <ul style="list-style-type: none">• The temporary directory must have at least 1 GB of free space.• You must have write permission to the directory.
<code>--user=username</code> <code>-U username</code>	<p>Specifies the dbadmin user name. By default, <code>scrutinize</code> uses the user name of the invoking user.</p>
<code>--version</code>	<p>Displays the version number of the Vertica server and the <code>scrutinize</code> version number.</p>
<code>--vsq1-off</code> <code>-v</code>	<p>Excludes Query and SystemTable tasks, which are used to connect to the database. This option can help you deal with problems that occur during an upgrade, and is typically used in the following cases:</p> <ul style="list-style-type: none">• Vertica is running but is slow to respond.• You haven't created a database but need help troubleshooting other cluster issues.
<code>-w</code> <code>--prompt-password</code>	<p><code>scrutinize</code> displays a DB Password prompt at the command line, where users must enter their database passwords.</p>

Limiting the Scope of `scrutinize` Data Collection

`scrutinize` provides several options that let you narrow scope of the data collection, to reduce output size and improve performance:

- [Limit the time span of collected data.](#)
- [Limit the amount of collected data.](#)
- [Limit collection to specific cluster nodes.](#)

- [Specify the type of content to collect.](#)
- [Exclude types of content.](#)

You can use these options singly or in combination, to achieve the desired level of granularity.

Limiting the Time Span of Collected Data

You can use `scrutinize --begin` and `--end` options together or separately to limit the time frame of diagnostic data that is collected from `vertica.log` and `editor.log`:

- `--begin` specifies to collect log data from the specified begin time to the specified end time or log end.
- `--end` specifies to collect data from the specified begin time or log start to the specified end time.

You can specify begin and end *time* arguments as offsets from the current time or as absolute times:

Offset time format:	'DdHhMmSs' For example: '3d2h1m'
Absolute time format:	'YYYY-MM-DD HH:MM:SS' For example: '2014-01-02 13:29:59'

For example:

```
$ scrutinize --begin='2014-01-01 00:00:00' --end='2014-01-07 23:59:59'  
$ scrutinize --begin='2014-01-01 00:00:00'  
$ scrutinize --end='2014-01-07 23:59:59'  
$ scrutinize --begin='2014-09-23 00:00:00' --end='1d''
```

Limiting the Amount of Collected Data

Two options let you limit how much data `scrutinize` collects:

<code>--log-limit=<i>limit</i></code> <code>-l <i>limit</i></code>	Limits how much data is collected from Vertica logs, where <i>limit</i> specifies, in gigabytes, how much log data to collect,
---	--

	<p>starting from the most recent log entry. By default, 1 GB of log data is collected.</p> <p>For example, the following command specifies to collect 4 GB of log data:</p> <pre>\$ scrutinize --log-limit=4</pre>
<pre>--include_gzlogs=num-files -z num-files</pre>	<p>Specifies to include <i>num-files</i> rotated log files (<i>vertica.log*.gz</i>) in the <i>scrutinize</i> output, where <i>num-files</i> can be one of the following:</p> <ul style="list-style-type: none"> • An integer specifies the number of rotated log files to collect. • <i>all</i> specifies to collect all rotated log files. <p>By default, <i>scrutinize</i> includes three rotated log files.</p> <p>For example the following command specifies to collect two rotated log files:</p> <pre>\$ scrutinize --include_gzlogs=2</pre>

Limiting Collection to Specific Cluster Nodes

By default, *scrutinize* collects data from all cluster nodes. You can specify that *scrutinize* collect from individual nodes in two ways:

<pre>--local_diags -s</pre>	<p>Specifies to collect diagnostics only from the host on which <i>scrutinize</i> was invoked.</p>
<pre>--hosts=host-List -n host-List</pre>	<p>Specifies to collect diagnostics only from the hosts specified in <i>host-List</i>, where <i>host-List</i> is a comma-separated list of IP addresses or host names.</p> <p>Example:</p> <pre>\$ scrutinize -n host_1,host_2,host_3</pre> <pre>\$ scrutinize --hosts=127.0.0.1,host_3,host_1</pre>

Specifying the Type of Content to Collect

`scrutinize` provides several options that let you specify the type of data to collect:

<code>--diag-dump</code>	Limits the collection to database design, system tables, and Data Collector tables. Use this option to collect data to analyze system performance.
<code>--diagnostics</code>	Limits the collection to log file data and output from commands that are run against Vertica and its host system. Use this option to collect data to evaluate unexpected behavior in your Vertica system.
<code>--type=<i>type</i></code> <code>-t <i>type</i></code>	Specifies the type of diagnostics collection to perform, where <i>type</i> can be one of the following arguments: <ul style="list-style-type: none">• <code>profiling</code>: Gather profiling data.• <code>context</code>: Gather summary information.
<code>--tasks=<i>tasks</i></code> <code>-T <i>tasks</i></code>	Specifies that <code>scrutinize</code> gather diagnostics on one or more tasks, as specified in a file or JSON list. This option is typically used together with the <code>--exclude</code> option. Note: Use this option only in consultation with Vertica Customer Support

Excluding Types of Content

`scrutinize` provides two options that let you specify types of data to exclude from its collection:

<code>--vsq1-off</code> <code>-v</code>	Excludes Query and SystemTable tasks, which are used to connect to the database. This option can help you deal with problems that occur during an upgrade, and is typically used in the following cases: <ul style="list-style-type: none">• Vertica is running but is slow to respond.• You haven't yet created a database but need help troubleshooting other cluster issues.
<code>--exclude=<i>tasks</i></code> <code>-X <i>tasks</i></code>	Excludes one or more types of tasks from the diagnostics collection, where <i>tasks</i> is a comma-separated list of the tasks to exclude.

Note: This option is typically used only in consultation with your Vertica Customer Support contact.

Specify the tasks to exclude with the following case-insensitive arguments :

- `all`: All default tasks
- `DC`: Data Collector tables
- `File`: Log files from the installation process, the database, and Administration Tools, such as `vertica.log`, `dbLog`, and `adminTools.log`
- `VerticaLog`: Vertica logs
- `CatalogObject`: Vertica catalog metadata, such as system configuration parameters
- `SystemTable`: Vertica system tables that contain information about system, resources, workload, and performance
- `Query`: Vertica metafunctions that use `vsql` to connect to the database, such as `EXPORT_CATALOG()`
- `Command`: Operating system information, such as the length of time that a node has been up

scrutinize Security

`scrutinize` supports password handling in the following ways:

- The `--password (-P)` option sets the database password as an argument to the `scrutinize` command.

Omitting this argument on a password-protected database returns a warning. Enclose the password with single quotes to protect passwords with special characters.

For example:

```
$ scrutinize -P '@passWord**'  
$ scrutinize --password='$password1*'
```

Use this option if the administrator account (default dbadmin) has password authentication.

- The `--prompt-password (-W)` option invokes a DB Password prompt at the command line, where users must enter their database password.
- Support for [VSQL_PASSWORD](#)

Troubleshooting scrutinize

The troubleshooting advice in this section can help you resolve common issues that you might encounter when using `scrutinize`.

Collection Time Is Too Slow

To speed up collection time, omit system tables when running an instance of `scrutinize`. Be aware that collecting from fewer nodes does not necessarily speed up the collection process.

Output Size Is Too Large

Output size depends on system table size and vertica log size.

To create a smaller `scrutinize` output, omit some system tables or truncate the vertica log. For more information, see [Narrowing the Scope of scrutinize Data Collection](#).

System Tables Not Collected on Databases with Password

Running `scrutinize` on a password-protected database might require you to supply a user name and password:

```
$ scrutinize -U username -P 'password'
```

Uploading scrutinize Results to Support

In most cases, your Customer Support contact provides arguments for you to run with the `scrutinize` script. These arguments include the URL that you supply with the `--url` or `--auth-upload` options.

Before you run `scrutinize` with an upload option:

- Install the `cURL` program installed in the path for the database administrator user who is running `scrutinize`.
- Verify each node in the cluster can make an HTTP or FTP connection directly to the Internet.

When you use the upload options:

- `scrutinize` does not collect results into a single `VerticaScrutinizetimestamp.zip` file.
- Each node posts the files directly to the specified URL:
 - A smaller, context file is posted first, which lets Customer Support review high-level information.
 - The larger, more complete diagnostics collection downloads at the end of the run.

Authenticated Upload

You can use the `--url` or the `--auth-upload` option to upload `scrutinize` output to Vertica Customer Support. Each option authenticates the upload in a different way:

- The `--url` upload method requires the URL to include a user name and password.
- When you use the `--auth-upload` option, `scrutinize` uploads your customer name from your Vertica Premium Edition license. Customer Support uses this information to verify your identity on receiving your uploaded file.

The `--auth-upload` argument requires a valid Vertica Premium Edition license. In both cases, the upload uses a URL or FTP address supplied by Vertica Customer Support.

Upload Examples

The following example shows a generic upload that invokes the `--url` option to gather all diagnostics on the cluster and posts the collected output to a support-provided URL:

```
$ scrutinize -U username -P 'password' -u 'ftp://username/password@customers.vertica.com/'
```

You can also use the authenticated upload method to gather the same information and your customer name:

```
$ scrutinize -U username -P 'password' --auth-upload "url"
```

Including Messages in scrutinize Output

The `scrutinize` option `--message (-m)` includes a message in the `scrutinize` output, in `reason.txt`. Messages are typically included in order to provide the following information:

- Reason for gathering/submitting diagnostics.
- Support-supplied case number and other issue-specific information, in order to help Vertica Customer Support identify your case and analyze the problem.

If no message is specified, `scrutinize` generates the default message 'Unknown reason for collection.'

You can set a message several ways:

- Supply a message string enclosed by double-quotation marks (`"`). For example:

```
$ scrutinize --message="re: case number #ABC-12345"
```

- Supply the path to a file enclosed by double quotation marks. `scrutinize` includes the file contents in its output file. For example:

```
$ scrutinize --message="/path/to/file"
```

- The `PROMPT` keyword opens an input stream. `scrutinize` reads input until you type a period (`.`) on a new line. This closes the input stream, and `scrutinize` writes the message to the collected output. For example:

```
$ scrutinize --message=PROMPT
Enter reason for collecting diagnostics; end with '.' on a line by itself:
Query performance degradation noticed around 9AM EST on Saturday
.
Vertica Scrutinize Report
-----
Result Dir:                /home/dbadmin/VerticaScrutinize.20131126083311
...
```

Database Identifiers

`scrutinize` generates a unique identifier for the database that it analyzes. This identifier is based on your Vertica license and the database creation time. `scrutinize` generates the same name for a given database each time it collects data on that database. It displays this name near the end of its standard output—in the following example, **PURPLE LEAD**:

```
Vertica Scrutinize Report
-----
Result Dir:                /home/dbadmin/VerticaScrutinize.20140206144429
...
Gathered diagnostics for
Customer: Vertica Systems, Inc.
Database designation: PURPLE LEAD
Timestamp: 20140206144429
...
```

You can use this identifier when you interact with Customer Support. It is particularly useful in the following cases:

- A cluster hosts multiple databases.
- Different clusters host databases with the same name.

Exporting a Catalog

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the [EXPORT_CATALOG](#) function in the SQL Reference Manual for details.

Exporting Profiling Data

The diagnostics audit script gathers system table contents, design, and planning objects from a running database and exports the data into a file named `./diag_dump_<timestamp>.tar.gz`, where `<timestamp>` denotes when you ran the script.

If you run the script without parameters, you will be prompted for a database password.

Syntax

```
/opt/vertica/scripts/collect_diag_dump.sh [ command... ]
```

Parameters

<i>command</i>	-U	User name, typically the database administrator account, dbadmin.
	-w	Database password.
	-c	Includes a compression analysis, resulting in a longer script execution time.

Example

The following command runs the audit script with all arguments:

```
$ /opt/vertica/scripts/collect_diag_dump.sh -U dbadmin -w password -c
```

Profiling Database Performance

You can profile database performance to evaluate how efficiently queries execute. For example, profiles can provide the following information:

- How much memory and how many threads each operator is allocated.
- How data flows through each operator at different points in time during query execution.
- Whether a query is network bound.

Profiling information can help you evaluate query performance and determine whether to rewrite the query. You can use profiling when considering projection design issues, such as segmentation and sort order.

The topics in this section focus on obtaining profile data via vsql statements. The Vertica Management Console also provides an easy-to-read [view of query profile data](#).

Profiling Categories

Vertica divides profiling data into three categories and captures this information in several system tables. You can query those tables for specific performance information, as shown in the following table:

Profiling Category	System Tables	Description of Profiled Data
Session	SESSION_PROFILES	General information about query execution on each node during the current session. For example, you can find out how many statements ran successfully and unsuccessfully, how many locks were granted and deadlocks encountered, and so on.
Queries	QUERY_PLAN_PROFILES QUERY_PROFILES	Query-specific information, such as query string and duration of execution, divided between two system tables: <ul style="list-style-type: none">• QUERY_PLAN_PROFILES: Real-time status for each query plan path.• QUERY_PROFILES: Query information.

Profiling Category	System Tables	Description of Profiled Data
Execution engine	EXECUTION_ENGINE_PROFILES	Information on execution engine performance.

Note: If execution engine profiling is disabled, Vertica saves no data in `EXECUTION_ENGINE_PROFILES`. However, if query and session profiling is disabled, Vertica saves some data in `QUERY_PROFILES`, `QUERY_PLAN_PROFILES`, and `SESSION_PROFILES`.

For each category, you can enable profiling globally, for the entire database, or only for the current session. For more information, see [Enabling and Disabling Profiling](#).

Enabling and Disabling Profiling

You can enable and disable profiling for a specific category globally (for the entire database) and the current session. Use [SHOW_PROFILING_CONFIG](#) to determine whether profiling is enabled. Output from this command shows:

- At what scope (global and session) profiling is enabled.
- For what [categories](#) profiling is enabled.

Note: If profiling is enabled globally for a given category, enabling or disabling session profiling has no effect on whether Vertica gathers profile data for that category. The session profiling setting enables profiling for a given category only when global profiling is disabled for the same category.

In the following example, [SHOW_PROFILING_CONFIG](#) shows that profiling is enabled globally (Global on) across all profiling categories—session, EE (execution engine), and query. For the current session, it is disabled (Session off) across all categories:

```
=> SELECT SHOW_PROFILING_CONFIG();
SHOW_PROFILING_CONFIG
-----
Session Profiling: Session off, Global on
EE Profiling:      Session off, Global on
Query Profiling:   Session off, Global on
(1 row)
```

Enable or Disable Global Profiling

```
ALTER DATABASE db-name SET profiling-category = {0 | 1}
```

where *profiling-category* is set to one of the following arguments:

Use this argument...	To specify...
GlobalSessionProfiling	Session profile data
GlobalQueryProfiling	Query data
GlobalEETProfiling	Execution engine data

For example, the following statement globally enables query profiling:

```
=> ALTER DATABASE mydb SET GlobalQueryProfiling = 1;
```

Enable or Disable Session Profiling

```
ENABLE_PROFILING( profiling-category )  
DISABLE_PROFILING( profiling-category )
```

where *profiling-category* is set to one of the following arguments:

Use this argument...	To specify...
session	Session profile data
query	Query data
ee	Execution engine data

For example, the following statement enables session-scoped profiling for the execution run of each query:

```
=> SELECT ENABLE_PROFILING('ee');  
   ENABLE_PROFILING  
-----  
   EE Profiling Enabled  
(1 row)
```

What to look for in query profiles

Profile data can show data skew, when some nodes are processing more data than others. The `rows_produced` counter in the system table `EXECUTION_ENGINE_PROFILES` shows how many rows have been processed by each operator. Comparing the `rows_produced` across all nodes for a given operator reveals if there is a data skew issue.

Note: Some profiling operators in `EXECUTION_ENGINE_PROFILES` are generic, such as `JOIN`. The `EXPLAIN`-generated query plan includes more details about the specific join that is executing.

Real-Time Profiling

With real-time profiling, you can monitor long-running queries while they execute.

Real-time profiling counters are available for all statements while they execute, including internal operations such as mergeout, recovery, and refresh. Unless you explicitly enable profiling using the keyword `PROFILE` on a specific SQL statement, or generally [enable profiling](#) for the database and/or the current session, profiling counters are unavailable after the statement completes.

Queries for real-time profiling data require a transaction ID. If the transaction executes multiple statements, the query also requires a statement ID to identify the desired statement. You obtain transaction and statement IDs the `SYSTEM_SESSIONS` system table from columns `transaction_id` and `statement_id`, respectively:

```
=> SELECT transaction_id, statement_id FROM SYSTEM_SESSIONS;
```

Profiling Counters

The `EXECUTION_ENGINE_PROFILES` system table contains available profiling counters for internal operations and user statements.

Useful counters include:

- Execution time (μ s)
- Rows produced

- Total merge phases
- Completed merge phases
- Current size of temp files (bytes)

You can view all available counters by querying [EXECUTION_ENGINE_PROFILES](#):

```
=> SELECT DISTINCT(counter_name) FROM EXECUTION_ENGINE_PROFILES;
```

To monitor the profiling counters, you can run a command like the following using a retrieved transaction ID (a000000000027):

```
=> SELECT * FROM execution_engine_profiles  
WHERE TO_HEX(transaction_id)='a000000000027'  
AND counter_name = 'execution time (us)'  
ORDER BY node_name, counter_value DESC;
```

The following example finds operators with the largest execution time on each node:

```
=> SELECT node_name, operator_name, counter_value execution_time_us FROM V_MONITOR.EXECUTION_ENGINE_  
PROFILES WHERE counter_name='execution time (us)' LIMIT 1 OVER(PARTITION BY node_name ORDER BY  
counter_value DESC);
```

node_name	operator_name	execution_time_us
v_vmart_node0001	Join	131906
v_vmart_node0002	Join	227778
v_vmart_node0003	NetworkSend	524080

(3 rows)

Query Plan Profiles

You can obtain query-specific data by profiling the query statement and evaluating the data that is captured in system tables [QUERY_PLAN_PROFILES](#) and [EXECUTION_ENGINE_PROFILES](#). For example, you can query [QUERY_PLAN_PROFILES](#) to determine how much time a query spends on each query plan operation. For details, see [Profiling Query Plans](#).

Linux watch Command

You can use the Linux `watch` command to monitor long-running queries at frequent intervals. Common use cases include:

- Observing executing operators within a query plan on each Vertica cluster node.
- Monitoring workloads that might be unbalanced among cluster nodes—for example, some nodes become idle while others are active. Such imbalances might be caused by data skews or by hardware issues.

In the following example, watch queries operators with the largest execution time on each node. The command specifies to re-execute the query each second:

```
watch -n 1 -d "vsq1 VMart -c\"SELECT node_name, operator_name, counter_value execution_time_us
FROM v_monitor.execution_engine_profiles WHERE counter_name='execution time (us)'
LIMIT 1 OVER(PARTITION BY node_name ORDER BY counter_value DESC);

Every 1.0s: vsq1 VMart -c"SELECT node_name, operator_name, counter_value execution_time_us FROM v_
monitor.execu... Thu Jan 21 15:00:44 2016

  node_name      | operator_name | execution_time_us
-----+-----+-----
v_vmart_node0001 | Root          |           110266
v_vmart_node0002 | UnionAll     |             38932
v_vmart_node0003 | Scan         |             22058
(3 rows)
```

Profiling Single Statements

To profile a single statement, prefix it with [PROFILE](#). You can profile a query (SELECT) statement, or any DML statement such as [INSERT](#), [UPDATE](#), [COPY](#), and [MERGE](#). The statement returns with a profile summary:

- Profile identifiers `transaction_id` and `statement_id`
- Initiator memory for the query
- Total memory required

For example:

```
=> PROFILE SELECT customer_name, annual_income FROM public.customer_dimension
  WHERE (customer_gender, annual_income) IN (SELECT customer_gender, MAX(annual_income)
  FROM public.customer_dimension GROUP BY customer_gender);
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996278193955 and
statement_id=1;
NOTICE 3557: Initiator memory for query: [on pool pool1: 996147 KB, minimum: 746280 KB]
NOTICE 5077: Total memory required by query: [996147 KB]
  customer_name | annual_income
-----+-----
```

```
Emily G. Vogel | 999998
James M. McNulty | 999979
(2 rows)
```

You can use the profile identifiers `transaction_id` and `statement_id` to obtain detailed profile information for this query from system tables [EXECUTION_ENGINE_PROFILES](#) and [QUERY_PLAN_PROFILES](#). For example:

```
=> SELECT path_id, path_line::VARCHAR(68), running_time FROM v_monitor.query_plan_profiles
WHERE transaction_id=45035996278193955 AND statement_id=1 ORDER BY path_id, path_line_index;
path_id | path_line | running_time
-----+-----+-----
1 | +-JOIN HASH [Semi] [Cost: 644, Rows: 25K] (PATH ID: 1) | 00:00:00.009077
1 | | Join Cond: (customer_dimension.customer_gender = VAL(2)) AND (cus |
1 | | Materialize at Output: customer_dimension.customer_name |
1 | | Execute on: Query Initiator |
2 | | +- Outer -> STORAGE ACCESS for customer_dimension [Cost: 149, Row | 00:00:00.008763
2 | | | Projection: public.customer_dimension_DBD_1_rep_VMartDesign |
2 | | | Materialize: customer_dimension.customer_gender, customer_d |
2 | | | Execute on: Query Initiator |
2 | | | Runtime Filters: (SIP1(HashJoin): customer_dimension.custom |
4 | | | +---> GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 159, Rows: 3] | 00:00:00.006566
4 | | | | Aggregates: max(customer_dimension.annual_income) |
4 | | | | Group By: customer_dimension.customer_gender |
4 | | | | Execute on: Query Initiator |
5 | | | | +---> STORAGE ACCESS for customer_dimension [Cost: 149, Rows: | 00:00:00.006943
5 | | | | | Projection: public.customer_dimension_DBD_1_rep_VMartDe |
5 | | | | | Materialize: customer_dimension.customer_gender, custom |
5 | | | | | Execute on: Query Initiator |
(17 rows)
```

Sample Views for Counter Information

The `EXECUTION_ENGINE_PROFILES` table contains the data for each profiling counter as a row within the table. For example, the execution time (us) counter is in one row, and the rows produced counter is in a second row. Since there are many different profiling counters, many rows of profiling data exist for each operator. Some sample views are installed by default to simplify the viewing of profiling counters.

Running scripts to create the sample views

The following script creates the `v_demo` schema and places the views in that schema.

```
/opt/vertica/scripts/demo_eeprofile_view.sql
```

Viewing counter values using the sample views

There is one view for each of the profiling counters to simplify viewing of a single counter value. For example, to view the execution time for all operators, issue the following command from the database:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us;
```

To view all counter values available for all profiled queries:

```
=> SELECT * FROM v_demo.eeprof_counters;
```

To select all distinct operators available for all profiled queries:

```
=> SELECT * FROM v_demo.eeprof_operators;
```

Combining sample views

These views can be combined:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us  
NATURAL LEFT OUTER JOIN v_demo.eeprof_rows_produced;
```

To view the execution time and rows produced for a specific transaction and `statement_id` ranked by execution time on each node:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank  
WHERE transaction_id=45035996273709699  
AND statement_id=1  
ORDER BY transaction_id, statement_id, node_name, rk;
```

To view the top five operators by execution time on each node:

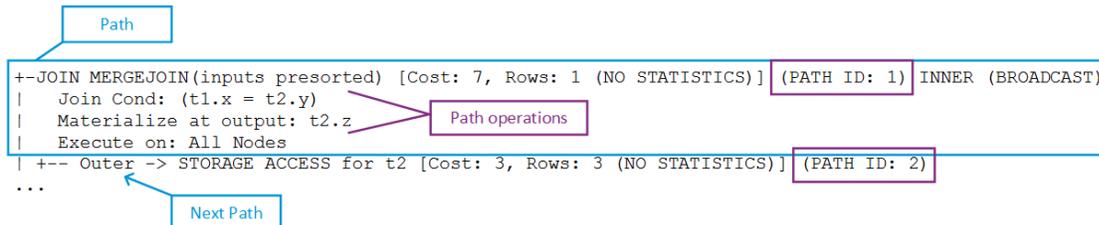
```
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank  
WHERE transaction_id=45035996273709699  
AND statement_id=1 AND rk<=5  
ORDER BY transaction_id, statement_id, node_name, rk;
```

Profiling Query Plans

To monitor real-time flow of data through a query plan, query system tables [EXECUTION_ENGINE_PROFILES](#) and [QUERY_PLAN_PROFILES](#). These tables provides data on how Vertica executed a query plan and its individual paths:

- [EXECUTION_ENGINE_PROFILES](#) summarizes query execution runs.
- [QUERY_PLAN_PROFILES](#) shows the real-time flow of data, and the time and resources consumed for each query plan path

Each query plan path has a unique ID, as shown in the following [EXPLAIN](#) output fragment.



Both tables provide path-specific data. For example, [QUERY_PLAN_PROFILES](#) provides high-level data for each path, which includes:

- Length of a query operation execution
- How much memory that path's operation consumed
- Size of data sent/received over the network

For example, you might observe that a `GROUP BY HASH` operation executed in 0.2 seconds using 100MB of memory.

Requirements

Real-time profiling minimally requires the ID of the transaction to monitor. If the transaction includes multiple statements, you also need the statement ID. You can get statement and transaction IDs by issuing [PROFILE](#) on the query to profile. You can then use these identifiers to query system tables [EXECUTION_ENGINE_PROFILES](#) and [QUERY_PLAN_PROFILES](#).

For more information, see [Profiling Single Statements](#).

Getting Query Plan Status for Small Queries

Real-time profiling counters, stored in the `EXECUTION_ENGINE_PROFILES` system table, are available for all currently executing statements—including internal operations, such as a mergeout.

Profiling counters are available after query execution has completed if any of the following conditions are true:

- The query was run via the `PROFILE <query>` command
- Systemwide profiling has been enabled through the `ENABLE_PROFILING()` function
- The query took longer than two seconds to run

Profiling counters are saved in the `EXECUTION_ENGINE_PROFILES` system table until the storage quota has been exceeded.

Here's an example:

1. Profile the query to get the `transaction_id` and `statement_id` from from `EXECUTION_ENGINE_PROFILES`; for example:

```
=> PROFILE SELECT * FROM t1 JOIN t2 ON t1.x = t2.y;
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996273955065
and statement_id=4;
NOTICE 3557: Initiator memory for query: [on pool general: 248544 KB, minimum: 248544 KB]
NOTICE 5077: Total memory required by query: [248544 KB]
 x | y | z
---+---+-----
 3 | 3 | three
(1 row)
```

2. Query the `QUERY_PLAN_PROFILES` system table.

Note: For best results, sort on the `transaction_id`, `statement_id`, `path_id`, and `path_line_index` columns.

```
=> SELECT ... FROM query_plan_profiles
WHERE transaction_id=45035996273955065 and statement_id=4;
ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

Getting Query Plan Status for Large Queries

Real-time profiling is designed to monitor large (long-running) queries. Take the following steps to monitor plans for large queries:

1. Get the statement and transaction IDs for the query plan you want to profile by querying the `CURRENT_SESSION` system table:

```
=> SELECT transaction_id, statement_id from current_session;
transaction_id | statement_id
-----+-----
45035996273955001 | 4
(1 row)
```

2. Run the query:

```
=> SELECT * FROM t1 JOIN t2 ON x=y JOIN ext on y=z;
```

3. Query the `QUERY_PLAN_PROFILES` system table, and sort on the `transaction_id`, `statement_id`, `path_id`, and `path_line_index` columns.

```
=> SELECT ... FROM query_plan_profiles WHERE transaction_id=45035996273955001 and statement_id=4
ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

You can also use the Linux `watch` command to monitor long-running queries (see [Real-Time Profiling](#)).

Example

The following series of commands creates a table for a long-running query and then runs the `QUERY_PLAN_PROFILES` system table:

```
=> CREATE TABLE longq(x int);
CREATE TABLE
=> COPY longq FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> 5
>> 6
>> 7
```

```
>> 8
>> 9
>> 10
>> \.
=> INSERT INTO longq SELECT f1.x+f2.x+f3.x+f4.x+f5.x+f6.x+f7.x
    FROM longq f1
    CROSS JOIN longq f2
    CROSS JOIN longq f3
    CROSS JOIN longq f4
    CROSS JOIN longq f5
    CROSS JOIN longq f6
    CROSS JOIN longq f7;
OUTPUT
-----
10000000
(1 row)
=> COMMIT;
COMMIT
```

Suppress query output on the terminal window by using the vsql `\o` command:

```
=> \o /home/dbadmin/longQprof
```

Query the new table:

```
=> SELECT * FROM longq;
```

Get the transaction and statement IDs:

```
=> SELECT transaction_id, statement_id from current_session;
transaction_id | statement_id
-----+-----
45035996273955021 |          4
(1 row)
```

Turn off the `\o` command so that Vertica continues to save query plan information to the file you specified. Alternatively, leave it on and examine the file after you query the `QUERY_PLAN_PROFILES` system table.

```
=> \o
```

Query the `QUERY_PLAN_PROFILES` system table;

```
=> SELECT
    transaction_id,
    statement_id,
    path_id,
    path_line_index,
    is_executing,
    running_time,
    path_line
FROM query_plan_profiles
WHERE transaction_id=45035996273955021 AND statement_id=4
```

```
ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

Improving Readability of QUERY_PLAN_PROFILES Output

Output from the [QUERY_PLAN_PROFILES](#) table can be very wide because of the `path_line` column. To facilitate readability, query `QUERY_PLAN_PROFILES` using one or more of the following options:

- Sort output by `transaction_id`, `statement_id`, `path_id`, and `path_line_index`:

```
=> SELECT ... FROM query_plan_profiles  
      WHERE ...  
      ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

- Use column aliases to decrease column width:

```
=> SELECT statement_id AS sid, path_id AS id, path_line_index AS order,  
       is_started AS start, is_completed AS end, is_executing AS exe,  
       running_time AS run, memory_allocated_bytes AS mem,  
       read_from_disk_bytes AS read, received_bytes AS rec,  
       sent_bytes AS sent, FROM query_plan_profiles  
      WHERE transaction_id=45035996273910558 AND statement_id=3  
      ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

- Use the `vsql \o` command to redirect [EXPLAIN](#) output to a file:

```
=> \o /home/dbadmin/long-queries  
=> EXPLAIN SELECT * FROM customer_dimension;  
=> \o
```

Managing Query Profile Data

Vertica retains data for queries until the storage quota for the table is exceeded, when it automatically purges the oldest queries to make room for new ones. You can clear profiled data by manually calling one of the following functions:

- [CLEAR_PROFILING\(\)](#) clears profiled data from memory. For example, the following command clears profiling for general query-run information, such as the query strings used and the duration of queries.

```
=> SELECT CLEAR_PROFILING('query');
```

- [CLEAR_DATA_COLLECTOR\(\)](#) clears all memory and disk records on the Data Collector tables and functions and resets collection statistics in the [V_MONITOR.DATA_COLLECTOR](#) system table.
- [FLUSH_DATA_COLLECTOR\(\)](#) waits until memory logs are moved to disk and then flushes the Data Collector, synchronizing the DataCollector log with the disk storage.

Configuring data retention policies

Vertica retains the historical data it gathers based on retention policies, which a superuser can configure. See [Retaining Monitoring Information](#).

Analyzing Suboptimal Query Plans

If profiling uncovers a suboptimal query, invoking one of the following functions might help:

- [ANALYZE_WORKLOAD](#) analyzes system information held in system tables and provides tuning recommendations that are based on a combination of statistics, system and data collector events, and database-table-projection design.
- [ANALYZE_STATISTICS](#) collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table or column.

You can also run your query through the Database Designer. See [Incremental Design](#).

Labeling Queries

To quickly identify queries for profiling and debugging purposes, include the [LABEL](#) hint.

LABEL hints are valid in the following statements:

- [DELETE](#)
- [INSERT](#)
- [MERGE](#)

- [SELECT](#)
- [UPDATE](#)
- [UNION](#): Valid in the UNION's first SELECT statement. Vertica ignores labels in subsequent SELECT statements.

For example:

```
SELECT /*+label(myselectquery)*/ COUNT(*) FROM t;  
INSERT /*+label(myinsertquery)*/ INTO t VALUES(1);
```

After you add a label to one or more queries, query the [QUERY_PROFILES](#) system table to see which queries ran with your supplied labels. The `QUERY_PROFILES` system table `IDENTIFIER` column returns the user-defined label that you previously assigned to a query. You can also obtain other query-specific data that can be useful for querying other system tables, such as transaction IDs.

For example:

```
=> SELECT identifier, query FROM query_profiles;  
  identifier | query  
-----+-----  
myselectquery | SELECT /*+label(myselectquery)*/ COUNT(*) FROM t;  
myinsertquery | INSERT /*+label(myinsertquery)*/ INTO t VALUES(1);  
myupdatequery | UPDATE /*+label(myupdatequery)*/ t SET a = 2 WHERE a = 1;  
mydeletequery | DELETE /*+label(mydeletequery)*/ FROM t WHERE a = 1;  
              | SELECT identifier, query from query_profiles;  
(5 rows)
```

About Locale

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate `GROUP BY` and `ORDER BY` clauses, joins, and the analytic `ORDER BY` clause.

The default locale for a Vertica database is `en_US@collation=binary` (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default `en_US@collation=binary` collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to `en_US_POSIX`. You can set the locale back to the default locale and collation by issuing the `vsq` meta-command `\locale`. For example:

```
=> set locale to '';
INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation=binary;
```

You can set locale through [ODBC](#), [JDBC](#), and [ADO.net](#).

Vertica locale specifications follow a subset of the [Unicode LDML](#) standard as implemented by the [ICU library](#).

Locale Handling in Vertica

The following sections describes how Vertica handles locale.

Session Locale

Locale is session-scoped and applies only to queries executed in that session. You cannot specify locale for individual queries. When you start a session it obtains its locale from the

configuration parameter `DefaultSessionLocale`.

Query Restrictions

The following restrictions apply when queries are run with locale other than the default `en_US@collation=binary`:

- When one or more of the left-side `NOT IN` columns is `CHAR` or `VARCHAR`, multi-column `NOT IN` subqueries are not supported. For example:

```
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
ERROR: Multi-expression NOT IN subquery is not supported because a left
hand expression could be NULL
```

Note: Even if columns `test.x` and `test.y` have a `NOT NULL` constraint, an error occurs.

- If the outer query contains a `GROUP BY` clause on a `CHAR` or `VARCHAR` column, correlated `HAVING` clause subqueries are not supported. In the following example, the `GROUP BY x` in the outer query causes the error:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x
    IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a' );
ERROR: subquery uses ungrouped column "t.x" from outer query
```

- Subqueries that use analytic functions in the `HAVING` clause are not supported. For example:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1) FROM test
    GROUP BY x HAVING x IN (SELECT MAX(x) FROM test);
ERROR: Analytics query with having clause expression that involves
aggregates and subquery is not supported
```

Collation and Projections

Projection data is sorted according to the default `en_US@collation=binary` collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by `col1` according to the binary collation:

```
=> CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY col1;
```

In such cases, `straße` and `strasse` are not stored near each other on disk.

Sorting by binary collation also means that sort optimizations do not work in locales other than binary. Vertica returns the following warning if you create tables or projections in a non-binary locale:

```
WARNING: Projections are always created and persisted in the default  
Vertica locale. The current locale is de_DE
```

Non-Binary Locale Input Handling

When the locale is non-binary, Vertica uses the [COLLATION](#) function to transform input to a binary string that sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

```
result_column_width = input_octet_width * CollationExpansion + 4
```

The default value of configuration parameter `CollationExpansion` is 5.

Character Data Type Handling

- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-binary locale.
- The maximum length parameter for VARCHAR and CHAR data type refers to the number of octets (bytes) that can be stored in that field and not number of characters. When using multi-byte UTF-8 characters, the fields must be sized to accommodate from 1 to 4 bytes per character, depending on the data.

Specifying Locale: Long Form

Vertica supports long forms that specify the `collation` keyword. Vertica extends long-form processing to accept collation arguments.

Syntax

[*Language*][*_script*][*_country*][*_variant*][*@collation-spec*]

Note: The following syntax options apply:

- Locale specification strings are case insensitive. For example, en_us and EN_US, are equivalent.
- You can substitute underscores with hyphens. For example: [-script]

Parameters

<i>Language</i>	A two- or three-letter lowercase code for a particular language. For example, Spanish is es English is en and French is fr. The two-letter language code uses the ISO-639 standard.
<i>_script</i>	An optional four-letter script code that follows the language code. If specified, it should be a valid script code as listed on the Unicode ISO 15924 Registry.
<i>_country</i>	A specific language convention within a generic language for a specific country or region. For example, French is spoken in many countries, but the currencies are different in each country. To allow for these differences among specific geographical, political, or cultural regions, locales are specified by two-letter, uppercase codes. For example, FR represents France and CA represents Canada. The two letter country code uses the ISO-3166 standard.
<i>_variant</i>	Differences may also appear in language conventions used within the same country. For example, the Euro currency is used in several European countries while the individual country's currency is still in circulation. To handle variations inside a language and country pair, add a third code, the variant code. The variant code is arbitrary and completely application-specific. ICU adds <code>_EURO</code> to its locale designations for locales that support the Euro currency. Variants can have any number of underscored key words. For example, <code>EURO_WIN</code> is a variant for the Euro currency on a Windows computer.

	<p>Another use of the variant code is to designate the Collation (sorting order) of a locale. For instance, the <code>es__TRADITIONAL</code> locale uses the traditional sorting order which is different from the default modern sorting of Spanish.</p>
<p><code>@collation-spec</code></p>	<p>Vertica only supports the keyword <code>collation</code>, as follows:</p> <pre>@collation=collation-type[;arg]...</pre> <p>Collation can specify one or more semicolon-delimited arguments, described below.</p> <p><i>collation-type</i> is set to one of the following values:</p> <ul style="list-style-type: none">• <code>big5han</code>: Pinyin ordering for Latin, big5 charset ordering for CJK characters (used in Chinese).• <code>dict</code>: For a dictionary-style ordering (such as in Sinhala).• <code>direct</code>: Hindi variant.• <code>gb2312/gb2312han</code>: Pinyin ordering for Latin, gb2312han charset ordering for CJK characters (used in Chinese).• <code>phonebook</code>: For a phonebook-style ordering (such as in German).• <code>pinyin</code>: Pinyin ordering for Latin and for CJK characters; that is, an ordering for CJK characters based on a character-by-character transliteration into a pinyin (used in Chinese).• <code>reformed</code>: Reformed collation (such as in Swedish).• <code>standard</code>: The default ordering for each language. For root it is [UCA] order; for each other locale it is the same as UCA (Unicode Collation Algorithm) ordering except for appropriate modifications to certain characters for that language. The following are additional choices for certain locales; they have effect only in certain locales.• <code>stroke</code>: Pinyin ordering for Latin, stroke order for CJK characters (used in Chinese) not supported.• <code>traditional</code>: For a traditional-style ordering (such as in Spanish).• <code>unihan</code>: Pinyin ordering for Latin, Unihan radical-stroke ordering

	<p>for CJK characters (used in Chinese) not supported.</p> <ul style="list-style-type: none"> • binary: Vertica default, providing UTF-8 octet ordering. <p>Notes:</p> <ul style="list-style-type: none"> • Collations might default to root, the ICU default collation. • Invalid values of the collation keyword and its synonyms do not cause an error. For example, the following does not generate an error. It simply ignores the invalid value: <pre>=> \locale en_GB@collation=xyz INFO 2567: Canonical locale: 'en_GB@collation=xyz' Standard collation: 'LEN' English (United Kingdom, collation=xyz)</pre> <p>For more about collation options, see Unicode Locale Data Markup Language (LDML).</p>
--	---

Collation Arguments

collation can specify one or more of the following arguments :

Parameter	Short form	Description
colstrength	S	<p>Sets the default strength for comparison. This feature is locale dependent.</p> <p>Set colstrength to one of the following:</p> <ul style="list-style-type: none"> • 1 primary: Ignores case and accents. Only primary differences are used during comparison—for example, a versus z. • 2 secondary: Ignores case. Only secondary and above differences are considered for comparison—for example, different accented forms of the same base letter such as a versus \u00E4. • 3 tertiary (default): Only tertiary differences and higher are considered for

Parameter	Short form	Description
		<p>comparison. Tertiary comparisons are typically used to evaluate case differences—for example, Z versus z.</p> <ul style="list-style-type: none"> • 4 quaternary: For example, used with Hiragana.
colAlternate	A	<p>Sets alternate handling for variable weights, as described in UCA, one of the following:</p> <ul style="list-style-type: none"> • non-ignorable N D • shifted S
colBackwards	F	<p>For Latin with accents, this parameter determines which accents are sorted. It sets the comparison for the second level to be backwards.</p> <p>Note: colBackwards is automatically set for French accents.</p> <p>Set colBackwards to one of the following:</p> <ul style="list-style-type: none"> • on 0: The normal UCA algorithm is used. • off X: All strings that are in Fast C or D normalization form (FCD) sort correctly, but others do not necessarily sort correctly. Set to off if the strings to be compared are in FCD.
colNormalization	N	<p>Set to one of the following:</p> <ul style="list-style-type: none"> • on 0: The normal UCA algorithm is used. • off X: All strings that are in Fast C or D normalization form (FCD) sort correctly, but others won't necessarily sort correctly. It should only be set off if the strings to be compared are in FCD.
colCaseLevel	E	<p>Set to one of the following:</p>

Parameter	Short form	Description
		<ul style="list-style-type: none"> • on 0: A level consisting only of case characteristics is inserted in front of tertiary level. To ignore accents but take cases into account, set strength to primary and case level to on. • off X: This level is omitted.
colCaseFirst	C	<p>Set to one of the following:</p> <ul style="list-style-type: none"> • upper U: Upper case sorts before lower case. • lower L: Lower case sorts before upper case. This is useful for locales that have already supported ordering but require different order of cases. It affects case and tertiary levels. • off short: Tertiary weights unaffected
colHiraganaQuaternary	H	<p>Controls special treatment of Hiragana code points on quaternary level, one of the following:</p> <ul style="list-style-type: none"> • on 0: Hiragana codepoints get lower values than all the other non-variable code points. The strength must be greater or equal than quaternary for this attribute to take effect. • off X: Hiragana letters are treated normally.
colNumeric	D	<p>If set to on, any sequence of Decimal Digits (General_Category = Nd in the [UCD]) is sorted at a primary level with its numeric value. For example, A-21 < A-123.</p>
variableTop	B	<p>Sets the default value for the variable top. All code points with primary weights less than or equal to the variable top will be considered variable, and are affected by the alternate</p>

Parameter	Short form	Description
		handling. For example, the following command sets variableTop to be HYPHEN (u2010) <pre>=> \locale en_US@colalternate=shifted;variabletop=u2010</pre>

Locale Processing Notes

- Incorrect locale strings are accepted if the prefix can be resolved to a known locale version.

For example, the following works because the language can be resolved:

```
=> \locale en_XX
INFO 2567: Canonical locale: 'en_XX'
Standard collation: 'LEN'
English (XX)
```

The following does not work because the language cannot be resolved:

```
=> \locale xx_XX
xx_XX: invalid locale identifier
```

- POSIX-type locales such as en_US.UTF-8 work to some extent in that the encoding part "UTF-8" is ignored.
- Vertica uses the icu4c-4_2_1 library to support basic locale/collation processing with some extensions. This does not currently meet [current standards for locale processing](https://tools.ietf.org/html/rfc5646) (<https://tools.ietf.org/html/rfc5646>).

Examples

Specify German locale as used in Germany (de), with phonebook-style collation:

```
=> \locale de_DE@collation=phonebook
INFO 2567: Canonical locale: 'de_DE@collation=phonebook'
Standard collation: 'KPHONEBOOK_LDE'
German (Germany, collation=Phonebook Sort Order)
Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)
```

Specify German locale as used in Germany (de), with phonebook-style collation and strength set to secondary:

```
=> \locale de_DE@collation=phonebook;colStrength=secondary
INFO 2567: Canonical locale: 'de_DE@collation=phonebook'
Standard collation: 'KPHONEBOOK_LDE_S2'
German (Germany, collation=Phonebook Sort Order)
Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)
```

Specifying Locale: Short Form

Vertica accepts locales in short form. You can use the short form to specify the locale and keyname pair/value names.

To determine the short form for a locale, type in the long form and view the last line of INFO, as follows:

```
\locale fr
INFO: Locale: 'fr'
INFO: French
INFO: français
INFO: Short form: 'LFR'
```

Examples

Specify en (English) locale:

```
\locale LEN
INFO: Locale: 'en'
INFO: English
INFO: Short form: 'LEN'
```

Specify German locale as used in Germany (de), with phonebook-style collation:

```
\locale LDE_KPHONEBOOK
INFO: Locale: 'de@collation=phonebook'
INFO: German (collation=Phonebook Sort Order)
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)
INFO: Short form: 'KPHONEBOOK_LDE'
```

Specify German locale as used in Germany (de), with phonebook-style collation:

```
\locale LDE_KPHONEBOOK_S2
INFO: Locale: 'de@collation=phonebook'
INFO: German (collation=Phonebook Sort Order)
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)
INFO: Short form: 'KPHONEBOOK_LDE_S2'
```

Supported Locales

The following are the supported locale strings for Vertica. Each locale can optionally have a list of key/value pairs (see [Specifying Locale: Long Form](#)).

Locale Name	Language or Variant	Region
af	Afrikaans	
af_NA	Afrikaans	Namibian Afrikaans
af_ZA	Afrikaans	South Africa
am	Ethiopic	
am_ET	Ethiopic	Ethiopia
ar	Arabic	
ar_AE	Arabic	United Arab Emirates
ar_BH	Arabic	Bahrain
ar_DZ	Arabic	Algeria
ar_EG	Arabic	Egypt
ar_IQ	Arabic	Iraq
ar_JO	Arabic	Jordan
ar_KW	Arabic	Kuwait
ar_LB	Arabic	Lebanon
ar_LY	Arabic	Libya
ar_MA	Arabic	Morocco
ar_OM	Arabic	Oman
ar_QA	Arabic	Qatar
ar_SA	Arabic	Saudi Arabia

Locale Name	Language or Variant	Region
ar_SD	Arabic	Sudan
ar_SY	Arabic	Syria
ar_TN	Arabic	Tunisia
ar_YE	Arabic	Yemen
as	Assamese	
as_IN	Assamese	India
az	Azerbaijani	
az_Cyrl	Azerbaijani	Cyrillic
az_Cyrl_AZ	Azerbaijani	Azerbaijan Cyrillic
az_Latn	Azerbaijani	Latin
az_Latn_AZ	Azerbaijani	Azerbaijan Latin
be	Belarusian	
be_BY	Belarusian	Belarus
bg	Bulgarian	
bg_BG	Bulgarian	Bulgaria
bn	Bengali	
bn_BD	Bengali	Bangladesh
bn_IN	Bengali	India
bo	Tibetan	
bo_CN	Tibetan	PR China
bo_IN	Tibetan	India
ca	Catalan	
ca_ES	Catalan	Spain

Locale Name	Language or Variant	Region
cs	Czech	
cs_CZ	Czech	Czech Republic
cy	Welsh	
cy_GB	Welsh	United Kingdom
da	Danish	
da_DK	Danish	Denmark
de	German	
de_AT	German	Austria
de_BE	German	Belgium
de_CH	German	Switzerland
de_DE	German	Germany
de_LI	German	Liechtenstein
de_LU	German	Luxembourg
el	Greek	
el_CY	Greek	Cyprus
el_GR	Greek	Greece
en	English	
en_AU	English	Australia
en_BE	English	Belgium
en_BW	English	Botswana
en_BZ	English	Belize
en_CA	English	Canada
en_GB	English	United Kingdom

Locale Name	Language or Variant	Region
en_HK	English	Hong Kong S.A.R. of China
en_IE	English	Ireland
en_IN	English	India
en_JM	English	Jamaica
en_MH	English	Marshall Islands
en_MT	English	Malta
en_NA	English	Namibia
en_NZ	English	New Zealand
en_PH	English	Philippines
en_PK	English	Pakistan
en_SG	English	Singapore
en_TT	English	Trinidad and Tobago
en_US	English	United States
en_US_POSIX	English	United States Posix
en_VI	English	U.S. Virgin Islands
en_ZA	English	Zimbabwe or South Africa
en_ZW	English	Zimbabwe
eo	Esperanto	
es	Spanish	
es_AR	Spanish	Argentina
es_BO	Spanish	Bolivia
es_CL	Spanish	Chile
es_CO	Spanish	Columbia

Locale Name	Language or Variant	Region
es_CR	Spanish	Costa Rica
es_DO	Spanish	Dominican Republic
es_EC	Spanish	Ecuador
es_ES	Spanish	Spain
es_GT	Spanish	Guatemala
es_HN	Spanish	Honduras
es_MX	Spanish	Mexico
es_NI	Spanish	Nicaragua
es_PA	Spanish	Panama
es_PE	Spanish	Peru
es_PR	Spanish	Puerto Rico
es_PY	Spanish	Paraguay
es_SV	Spanish	El Salvador
es_US	Spanish	United States
es_UY	Spanish	Uruguay
es_VE	Spanish	Venezuela
et	Estonian	
et_EE	Estonian	Estonia
eu	Basque	Spain
eu_ES	Basque	Spain
fa	Persian	
fa_AF	Persian	Afghanistan
fa_IR	Persian	Iran

Locale Name	Language or Variant	Region
fi	Finnish	
fi_FI	Finnish	Finland
fo	Faroese	
fo_FO	Faroese	Faroe Islands
fr	French	
fr_BE	French	Belgium
fr_CA	French	Canada
fr_CH	French	Switzerland
fr_FR	French	France
fr_LU	French	Luxembourg
fr_MC	French	Monaco
fr_SN	French	Senegal
ga	Gaelic	
ga_IE	Gaelic	Ireland
gl	Gallegan	
gl_ES	Gallegan	Spain
gsw	German	
gsw_CH	German	Switzerland
gu	Gujurati	
gu_IN	Gujurati	India
gv	Manx	
gv_GB	Manx	United Kingdom
ha	Hausa	

Locale Name	Language or Variant	Region
ha_Latn	Hausa	Latin
ha_Latn_GH	Hausa	Ghana (Latin)
ha_Latn_NE	Hausa	Niger (Latin)
ha_Latn_NG	Hausa	Nigeria (Latin)
haw	Hawaiian	Hawaiian
haw_US	Hawaiian	United States
he	Hebrew	
he_IL	Hebrew	Israel
hi	Hindi	
hi_IN	Hindi	India
hr	Croatian	
hr_HR	Croatian	Croatia
hu	Hungarian	
hu_HU	Hungarian	Hungary
hy	Armenian	
hy_AM	Armenian	Armenia
hy_AM_REVISED	Armenian	Revised Armenia
id	Indonesian	
id_ID	Indonesian	Indonesia
ii	Sichuan	
ii_CN	Sichuan	Yi
is	Icelandic	
is_IS	Icelandic	Iceland

Locale Name	Language or Variant	Region
it	Italian	
it_CH	Italian	Switzerland
it_IT	Italian	Italy
ja	Japanese	
ja_JP	Japanese	Japan
ka	Georgian	
ka_GE	Georgian	Georgia
kk	Kazakh	
kk_Cyrl	Kazakh	Cyrillic
kk_Cyrl_KZ	Kazakh	Kazakhstan (Cyrillic)
kl	Kalaallisut	
kl_GL	Kalaallisut	Greenland
km	Khmer	
km_KH	Khmer	Cambodia
kn	Kannada	
kn-IN	Kannada	India
ko	Korean	
ko_KR	Korean	Korea
kok	Konkani	
kok_IN	Konkani	India
kw	Cornish	
kw_GB	Cornish	United Kingdom
lt	Lithuanian	

Locale Name	Language or Variant	Region
lt_LT	Lithuanian	Lithuania
lv	Latvian	
lv_LV	Latvian	Latvia
mk	Macedonian	
mk_MK	Macedonian	Macedonia
ml	Malayalam	
ml_IN	Malayalam	India
mr	Marathi	
mr_IN	Marathi	India
ms	Malay	
ms_BN	Malay	Brunei
ms_MY	Malay	Malaysia
mt	Maltese	
mt_MT	Maltese	Malta
nb	Norwegian Bokml	
nb_NO	Norwegian Bokml	Norway
ne	Nepali	
ne_IN	Nepali	India
ne_NP	Nepali	Nepal
nl	Dutch	
nl_BE	Dutch	Belgium
nl_NL	Dutch	Netherlands
nn	Norwegian nynorsk	

Locale Name	Language or Variant	Region
nn_NO	Norwegian nynorsk	Norway
om	Oromo	
om_ET	Oromo	Ethiopia
om_KE	Oromo	Kenya
or	Oriya	
or_IN	Oriya	India
pa	Punjabi	
pa_Arab	Punjabi	Arabic
pa_Arab_PK	Punjabi	Pakistan (Arabic)
pa_Guru	Punjabi	Gurmukhi
pa_Guru_IN	Punjabi	India (Gurmukhi)
pl	Polish	
pl_PL	Polish	Poland
ps	Pashto	
ps_AF	Pashto	Afghanistan
pt	Portuguese	
pt_BR	Portuguese	Brazil
pt_PT	Portuguese	Portugal
ro	Romanian	
ro_MD	Romanian	Moldavia
ro_RO	Romanian	Romania
ru	Russian	
ru_RU	Russian	Russia

Locale Name	Language or Variant	Region
ru_UA	Russian	Ukraine
si	Sinhala	
si_LK	Sinhala	Sri Lanka
sk	Slovak	
sk_SK	Slovak	Slovakia
sl	Slovenian	
sl_SL	Slovenian	Slovenia
so	Somali	
so_DJ	Somali	Djibouti
so_ET	Somali	Ethiopia
so_KE	Somali	Kenya
so_SO	Somali	Somalia
sq	Albanian	
sq_AL	Albanian	Albania
sr	Serbian	
sr_Cyrl	Serbian	Cyrillic
sr_Cyrl_BA	Serbian	Bosnia and Herzegovina (Cyrillic)
sr_Cyrl_ME	Serbian	Montenegro (Cyrillic)
sr_Cyrl_RS	Serbian	Serbia (Cyrillic)
sr_Latn	Serbian	Latin
sr_Latn_BA	Serbian	Bosnia and Herzegovina (Latin)
sr_Latn_ME	Serbian	Montenegro (Latin)
sr_Latn_RS	Serbian	Serbia (Latin)

Locale Name	Language or Variant	Region
sv	Swedish	
sv_FI	Swedish	Finland
sv_SE	Swedish	Sweden
sw	Swahili	
sw_KE	Swahili	Kenya
sw_TZ	Swahili	Tanzania
ta	Tamil	
ta_IN	Tamil	India
te	Telugu	
te_IN	Telugu	India
th	Thai	
th_TH	Thai	Thailand
ti	Tigrinya	
ti_ER	Tigrinya	Eritrea
ti_ET	Tigrinya	Ethiopia
tr	Turkish	
tr_TR	Turkish	Turkey
uk	Ukrainian	
uk_UA	Ukrainian	Ukraine
ur	Urdu	
ur_IN	Urdu	India
ur_PK	Urdu	Pakistan
uz	Uzbek	

Locale Name	Language or Variant	Region
uz_Arab	Uzbek	Arabic
uz_Arab_AF	Uzbek	Afghanistan (Arabic)
uz_Cryl	Uzbek	Cyrillic
uz_Cryl_UZ	Uzbek	Uzbekistan (Cyrillic)
uz_Latin	Uzbek	Latin
us_Latin_UZ		Uzbekistan (Latin)
vi	Vietnamese	
vi_VN	Vietnamese	Vietnam
zh	Chinese	
zh_Hans	Chinese	Simplified Han
zh_Hans_CN	Chinese	China (Simplified Han)
zh_Hans_HK	Chinese	Hong Kong SAR China (Simplified Han)
zh_Hans_MO	Chinese	Macao SAR China (Simplified Han)
zh_Hans_SG	Chinese	Singapore (Simplified Han)
zh_Hant	Chinese	Traditional Han
zh_Hant_HK	Chinese	Hong Kong SAR China (Traditional Han)
zh_Hant_MO	Chinese	Macao SAR China (Traditional Han)
zh_Hant_TW	Chinese	Taiwan (Traditional Han)
zu	Zulu	
zu_ZA	Zulu	South Africa

Locale and UTF-8 Support

Vertica supports Unicode Transformation Format-8, or UTF8, where 8 equals 8-bit. UTF-8 is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard. Initial encoding of byte codes and character assignments for UTF-8 coincides with ASCII. Thus, UTF8 requires little or no change for software that handles ASCII but preserves other values.

Vertica database servers expect to receive all data in UTF-8, and Vertica outputs all data in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. JDBC and ADO.NET APIs operate on data in UTF-16. Client drivers automatically convert data to and from UTF-8 when sending to and receiving data from Vertica using API calls. The drivers do not transform data loaded by executing a [COPY](#) or [COPY LOCAL](#) statement.

UTF-8 String Functions

The following string functions treat VARCHAR arguments as UTF-8 strings (when USING OCTETS is not specified) regardless of locale setting.

String function	Description
LOWER	Returns a VARCHAR value containing the argument converted to lowercase letters.
UPPER	Returns a VARCHAR value containing the argument converted to uppercase letters.
INITCAP	Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.
INSTR	Searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.
SPLIT_PART	Splits string on the delimiter and returns the location of the beginning of the given field (counting from one).
POSITION	Returns an integer value representing the character location of a specified substring with a string (counting from one).

String function	Description
STRPOS	Returns an integer value representing the character location of a specified substring within a string (counting from one).

Locale-Aware String Functions

Vertica provides string functions to support internationalization. Unless otherwise specified, these string functions can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of characters. Specify this information by adding the parameter `USING OCTETS` and `USING CHARACTERS` (default) to the function.

The following table lists all string functions that are locale-aware:

String function	Description
BTRIM	Removes the longest string consisting only of specified characters from the start and end of a string.
CHARACTER_LENGTH	Returns an integer value representing the number of characters or octets in a string.
GREATEST	Returns the largest value in a list of expressions.
GREATESTB	Returns its greatest argument, using binary ordering, not UTF-8 character ordering.
INITCAP	Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.
INSTR	Searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.
LEAST	Returns the smallest value in a list of expressions.
LEASTB	Returns its least argument, using binary ordering, not UTF-8 character ordering.
LEFT	Returns the specified characters from the left side of a string.
LENGTH	Takes one argument as an input and returns returns an integer value

String function	Description
	representing the number of characters in a string.
LTRIM	Returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).
OVERLAY	Returns a VARCHAR value representing a string having had a substring replaced by another string.
OVERLAYB	Returns an octet value representing a string having had a substring replaced by another string.
REPLACE	replaces all occurrences of characters in a string with another set of characters.
RIGHT	Returns the <i>length</i> right-most characters of string.
SUBSTR	Returns a VARCHAR value representing a substring of a specified string.
SUBSTRB	Returns a byte value representing a substring of a specified string.
SUBSTRING	Given a value, a position, and an optional length, returns a value representing a substring of the specified string at the given position.
TRANSLATE	Replaces individual characters in <i>string_to_replace</i> with other characters.
UPPER	Returns a VARCHAR value containing the argument converted to uppercase letters.

Appendix: Binary File Formats

This appendix presents the binary file format details required to load files into the Vertica database with the COPY statement. Binary files must adhere to the required format.

Creating Native Binary Format Files

Using COPY to load data with the NATIVE parser requires that the input data files adhere to the format described below. All NATIVE files must contain:

- A file signature.
- A set of column size definitions.
- The rows of data.

Note: You cannot mix Binary and ASCII source files in the same COPY statement.

File Signature

The first part of a NATIVE binary file consists of a file signature. The contents of the signature are fixed, and listed in the following table.

Byte Offset	0	1	2	3	4	5	6	7	8	9	10
Hex Value	4E	41	54	49	56	45	0A	FF	0D	0A	00
Text Literals	N	A	T	I	V	E	E'\n'	E'\317'	E'\r'	E'\n'	E'\000'

The purpose of the required signature is to ensure that the file has neither been corrupted by a non-8-bit file transfer, nor stripped of carriage returns, linefeeds, or null values. If the signature is intact, Vertica determines that the file has not been corrupted.

Column Definitions

Following the file signature, the file must define the widths of each column in the file as follows.

Byte Offset	Length (bytes)	Description	Comments
11	4	Header area length	32-bit integer in little-endian format that contains the length in bytes of remaining in the header, not including itself. This is the number of bytes from the end of this value to the start of the row data.
15	2	NATIVE file version	16-bit integer in little-endian format containing the version number of the NATIVE file format. The only valid value is currently 1. Future changes to the format could be assigned different version numbers to maintain backward compatibility.
17	1	Filler	Always 0.
18	2	Number of columns	16-bit integer in little-endian format that contains the number of columns in each row in the file.
20+	4 bytes for each column of data in the table	Column widths	Array of 32-bit integers in little-endian format that define the width of each column in the row. Variable-width columns have a value of -1 (0xFF 0xFF 0xFF 0xFF).

Note: All integers in NATIVE files are in **little-endian format** (least significant byte first).

The width of each column is determined by the data type it contains. The following table explains the column width needed for each data type, along with the data encoding.

Data Type	Length (bytes)	Column Content
INTEGER	1, 2, 4, 8	8-, 16-, 32-, and 64-bit integers are supported. All multi-byte values are stored in little-endian format. Note: All values for a column must be the width you

Data Type	Length (bytes)	Column Content
		specify here. If you set the length of an INTEGER column to be 4 bytes, then all of the values you supply for that column must be 32-bit integers.
BOOLEAN	1	0 for false, 1 for true.
FLOAT	8	Encoded in IEEE-754 format.
CHAR	User-specified	<ul style="list-style-type: none"> • Strings shorter than the specified length must be right-padded with spaces (E'\040'). • Strings are not null-terminated. • Character encoding is UTF-8. • UTF-8 strings can contain multi-byte characters. Therefore, number of characters in the string may not equal the number of bytes.
VARCHAR	4-byte integer (length) + data	<p>The column width for a VARCHAR column is always -1 to signal that it contains variable-length data.</p> <ul style="list-style-type: none"> • Each VARCHAR column value starts with a 32-bit integer that contains the number of bytes in the string. • The string must not be null-terminated. • Character encoding must be UTF-8. • Remember that UTF-8 strings can contain multi-byte characters. Therefore, number of characters in the string may not equal the number of bytes.
DATE	8	64-bit integer in little-endian format containing the Julian day since Jan 01 2000 (J2451545)
TIME	8	64-bit integer in little-endian format containing the number of microseconds since midnight in the UTC time zone.
TIMETZ	8	64-bit value where

Data Type	Length (bytes)	Column Content
		<ul style="list-style-type: none"> Upper 40 bits contain the number of microseconds since midnight. Lower 24 bits contain time zone as the UTC offset in microseconds calculated as follows: Time zone is logically from -24hrs to +24hrs from UTC. Instead it is represented here as a number between 0hrs to 48hrs. Therefore, 24hrs should be added to the actual time zone to calculate it. <p>Each portion is stored in little-endian format (5 bytes followed by 3 bytes).</p>
TIMESTAMP	8	64-bit integer in little-endian format containing the number of microseconds since Julian day: Jan 01 2000 00:00:00.
TIMESTAMPPTZ	8	A 64-bit integer in little-endian format containing the number of microseconds since Julian day: Jan 01 2000 00:00:00 in the UTC timezone.
INTERVAL	8	64-bit integer in little-endian format containing the number of microseconds in the interval.
BINARY	User-specified	Similar to CHAR. The length should be specified in the file header in the <i>Field Lengths</i> entry for the field. The field in the record must contain <i>length</i> number of bytes. If the value is smaller than the specified length, the remainder should be filled with nulls (E'\000').
VARBINARY	4-byte integer + data	Stored just like VARCHAR but data is interpreted as bytes rather than UTF-8 characters.
NUMERIC	(precision, scale) (precision ÷ 19 + 1) × 8 rounded up	A constant-length data type. Length is determined by the precision, assuming that a 64-bit unsigned integer can store roughly 19 decimal digits. The data consists of a sequence of 64-bit integers, each stored in little-endian format, with the most significant integer first. Data in the integers is stored in base 2 ⁶⁴ . 2's complement is used for

Data Type	Length (bytes)	Column Content
		<p>negative numbers.</p> <p>If there is a scale, then the numeric is stored as $\text{numeric} \times 10^{\text{scale}}$; that is, all real numbers are stored as integers, ignoring the decimal point. It is required that the scale matches that of the target column in the dataanchor table. Another option is to use FILLER columns to coerce the numeric to the scale of the target column.</p>

Row Data

Following the file header is a sequence of records that contain the data for each row of data. Each record starts with a header:

Length (bytes)	Description	Comments
4	Row length	<p>A 32-bit integer in little-endian format containing the length of the row's data in bytes. It includes the size of data only, not the header.</p> <p>Note: The number of bytes in each row can vary not only because of variable-length data, but also because columns containing NULL values do not have any data in the row. If column 3 has a NULL value, then column 4's data immediately follows the end of column 2's data. See the next</p>
Number of columns ÷ 8 rounded up (<code>CEILING (NumFields / (sizeof (uint8) * 8));</code>)	Null value bit field	<p>A series of bytes whose bits indicate whether a column contains a NULL. The most significant bit of the first byte indicates whether the first column in this row contains a NULL, the next most significant bit indicates whether the next column contains a NULL, and so on. If a bit is 1 (true) then the column contains a NULL, and there is no value for the column in the data for the row.</p>

Following the record header is the column values for the row. There is no separator characters for these values. Their location in the row of data is calculated based on where the previous column's data ended. Most data types have a fixed width, so their location is easy to

determine. Variable-width values (such as VARCHAR and VARBINARY) start with a count of the number of bytes the value contains.

See the table in the previous section for details on how each data type's value is stored in the row's data.

Example

The example below demonstrates creating a table and loading a NATIVE file that contains a single row of data. The table contains all possible data types.

```
=> CREATE TABLE allTypes (INTCOL INTEGER,
                           FLOATCOL FLOAT,
                           CHARCOL CHAR(10),
                           VARCHARCOL VARCHAR,
                           BOOLCOL BOOLEAN,
                           DATECOL DATE,
                           TIMESTAMPCOL TIMESTAMP,
                           TIMESTAMPTZCOL TIMESTAMPTZ,
                           TIMECOL TIME,
                           TIMETZCOL TIMETZ,
                           VARBINCOL VARBINARY,
                           BINCOL BINARY,
                           NUMCOL NUMERIC(38,0),
                           INTERVALCOL INTERVAL
                           );
=> COPY allTypes FROM '/home/dbadmin/allTypes.bin' NATIVE DIRECT;
=> \pset expanded
Expanded display is on.
=> SELECT * from allTypes;
-[ RECORD 1 ]-----
INTCOL      | 1
FLOATCOL    | -1.11
CHARCOL     | one
VARCHARCOL  | ONE
BOOLCOL     | t
DATECOL     | 1999-01-08
TIMESTAMPCOL | 1999-02-23 03:11:52.35
TIMESTAMPTZCOL | 1999-01-08 07:04:37-05
TIMECOL     | 07:09:23
TIMETZCOL   | 15:12:34-04
VARBINCOL   | \253\315
BINCOL      | \253
NUMCOL      | 1234532
INTERVALCOL | 03:03:03
```

The content of the `allTypes.bin` file appears below as a raw hex dump:

```
4E 41 54 49 56 45 0A FF 0D 0A 00 3D 00 00 00 01 00 00 0E 00
08 00 00 00 08 00 00 00 0A 00 00 00 FF FF FF FF 01 00 00 00
08 00 00 00 08 00 00 00 08 00 00 00 08 00 00 00 08 00 00 00
FF FF FF FF 03 00 00 00 18 00 00 00 08 00 00 00 73 00 00 00
00 00 01 00 00 00 00 00 00 00 C3 F5 28 5C 8F C2 F1 BF 6F 6E
65 20 20 20 20 20 20 20 03 00 00 00 4F 4E 45 01 9A FE FF FF
```

```
FF FF FF FF 30 85 B3 4F 7E E7 FF FF 40 1F 3E 64 E8 E3 FF FF
C0 2E 98 FF 05 00 00 00 D0 97 01 80 F0 79 F0 10 02 00 00 00
AB CD AB CD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 64 D6 12 00 00 00 00 00 C0 47 A3 8E 02 00 00 00
```

The following table breaks this file down into each of its components, and describes the values it contains.

Hex Values	Description	Value
4E 41 54 49 56 45 0A FF 0D 0A 00	Signature	NATIVE\n\317\r\n\000
3D 00 00 00	Header area length	61 bytes
01 00	Native file format version	Version 1
00	Filler value	0
0E 00	Number of columns	14 columns
08 00 00 00	Width of column 1 (INTEGER)	8 bytes
08 00 00 00	Width of column 2 (FLOAT)	8 bytes
0A 00 00 00	Width of column 3 (CHAR(10))	10 bytes
FF FF FF FF	Width of column 4 (VARCHAR)	-1 (variable width column)
01 00 00 00	Width of column 5 (BOOLEAN)	1 bytes
08 00 00 00	Width of column 6 (DATE)	8 bytes
08 00 00 00	Width of column 7 (TIMESTAMP)	8 bytes

Hex Values	Description	Value
08 00 00 00	Width of column 8 (TIMESTAMPTZ)	8 bytes
08 00 00 00	Width of column 9 (TIME)	8 bytes
08 00 00 00	Width of column 10 (TIMETZ)	8 bytes
FF FF FF FF	Width of column 11 (VARBINARY)	-1 (variable width column)
03 00 00 00	Width of column 12 (BINARY)	3 bytes
18 00 00 00	Width of column 13 (NUMERIC)	24 bytes. The size is calculated by dividing 38 (the precision specified for the numeric column) by 19 (the number of digits each 64-bit chunk can represent) and adding 1. $38 \div 19 + 1 = 3$. then multiply by eight to get the number of bytes needed. $3 \times 8 = 24$ bytes.
08 00 00 00	Width of column 14 (INTERVAL). last portion of the header section.	8 bytes
73 00 00 00	Number of bytes of data for the first row. this is the start of the first row of data.	115 bytes
00 00	Bit field for the null values contained in the first row of data	The row contains no null values.
01 00 00 00 00 00 00 00	Value for 64-bit INTEGER column	1
C3 F5 28 5C	Value for the	-1.11

Hex Values	Description	Value
8F C2 F1 BF	FLOAT column	
6F 6E 65 20 20 20 20 20 20 20	Value for the CHAR(10) column	"one " (padded With 7 spaces to fill the full 10 characters for the column)
03 00 00 00	The number of bytes in the following VARCHAR value.	3 bytes
4F 4E 45	The value for the VARCHAR column	"ONE"
01	The value for the BOOLEAN column	True
9A FE FF FF FF FF FF FF	The value for the DATE column	1999-01-08
30 85 B3 4F 7E E7 FF FF	The value for the TIMESTAMP column	1999-02-23 03:11:52.35
40 1F 3E 64 E8 E3 FF FF	The value for the TIMESTAMPTZ column	1999-01-08 07:04:37-05
C0 2E 98 FF 05 00 00 00	The value for the TIME column	07:09:23
D0 97 01 80 F0 79 F0 10	The value for the TIMETZ column	15:12:34-05
02 00 00 00	The number of bytes in the following VARBINARY value	2 bytes
AB CD	The value for the VARBINARY column	Binary data (\253\315 as octal values)

Hex Values	Description	Value
AB CD	The value for the BINARY column	Binary data (\253\315 as octal values)
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 64 D6 12 00 00 00 00 00	The value for the NUMERIC column	1234532
C0 47 A3 8E 02 00 00 00	The value for the INTERVAL column	03:03:03

See Also

- [COPY](#)

Analyzing Data

Welcome to the Analyzing Data Guide! This guide explains how to query and analyze data in your Vertica database.

Queries

Queries are database operations that retrieve data from one or more tables or views. In Vertica, the top-level SELECT statement is the query, and a query nested within another SQL statement is called a subquery.

Vertica is designed to run the same SQL standard queries that run on other databases. However, there are some differences between Vertica queries and queries used in other relational database management systems.

The Vertica [transaction model](#) is different from the SQL standard in a way that has a profound effect on query performance. You can:

- Run a query on a static backup of the database from any specific date and time. Doing so avoids holding locks or blocking other database operations.
- Use a subset of the standard SQL isolation levels and access modes (read/write or read-only) for a user session.

In Vertica, the primary structure of a SQL query is its statement. Each statement ends with a semicolon, and you can write multiple queries separated by semicolons; for example:

```
=> CREATE TABLE t1( ..., date_col date NOT NULL, ...);  
=> CREATE TABLE t2(..., state VARCHAR NOT NULL, ...);
```

Multiple Instances of Dimension Tables in the FROM Clause

The same dimension table can appear multiple times in a query's FROM clause, using different aliases. For example:

```
=> SELECT * FROM fact, dimension d1, dimension d2  
WHERE fact.fk = d1.pk  
AND  
fact.name = d2.name;
```

Historical Queries

Vertica supports querying historical data for individual SELECT statements.

Syntax

Vertica can run a query from a backup of the database taken at a specific date and time or at a specific epoch. The syntax is:

```
AT TIME 'timestamp' select...  
AT EPOCH epoch_number select...  
AT EPOCH LATEST select...
```

The command queries all data in the database up to and including the specified epoch or the epoch representing the specified date and time, without holding a lock or blocking write operations. The specified `TIMESTAMP` and `epoch_number` values must be greater than or equal to the Ancient History Mark epoch.

Parameters

AT EPOCH LATEST	Queries all committed data in the database up to, but not including, the current epoch.
AT EPOCH <i>epoch_number</i>	Queries all data in the database up to and including the specified epoch without holding a lock or blocking write operations.
AT TIME ' <i>timestamp</i> '	Queries all committed data in the database up to the time stamp specified. <code>AT TIME 'timestamp'</code> queries are resolved to the next epoch boundary before being evaluated.

Historical queries are useful because they access data in past epochs only. Historical queries do not need to hold table locks or block write operations because they do not return the absolute latest data. Their content is private to the transaction and valid only for the length of the transaction.

Historical queries behave in the same manner regardless of transaction isolation level. Historical queries observe only committed data, even excluding updates made by the current transaction, unless those updates are to a temporary table.

Be aware that there is only one backup of the logical schema. This means that any changes you make to the schema are reflected across all epochs. If, for example, you add a new column to a table and you specify a default value for the column, all historical epochs display the new column and its default value.

See Also

- [Transactions](#)
- [AboutQueryExecution](#)

Temporary Tables

You can use the `CREATE TEMPORARY TABLE` statement to implement certain queries using multiple steps:

1. Create one or more temporary tables.
2. Execute queries and store the result sets in the temporary tables.
3. Execute the main query using the temporary tables as if they were a normal part of the logical schema.

See [CREATE TEMPORARY TABLE](#) in the SQL Reference Manual for details.

SQL Queries

All DML (Data Manipulation Language) statements can contain queries. This section introduces some of the query types in Vertica, with additional details in later sections.

Note: Many of the examples in this chapter use the [VMart schema](#). For information about other Vertica-supplied queries, see the [Getting Started](#).

Simple Queries

Simple queries contain a query against one table. Minimal effort is required to process the following query, which looks for product keys and SKU numbers in the product table:

```
=> SELECT product_key, sku_number FROM public.product_dimension;
product_key | sku_number
-----+-----
```

```
43      | SKU-#129
87      | SKU-#250
42      | SKU-#125
49      | SKU-#154
37      | SKU-#107
36      | SKU-#106
86      | SKU-#248
41      | SKU-#121
88      | SKU-#257
40      | SKU-#120
(10 rows)
```

Joins

Joins use a relational operator that combines information from two or more tables. The query's `ON` clause specifies how tables are combined, such as by matching foreign keys to primary keys. In the following example, the query requests the names of stores with transactions greater than 70 by joining the store key ID from the store schema's sales fact and sales tables:

```
=> SELECT store_name, COUNT(*) FROM store.store_sales_fact
      JOIN store.store_dimension ON store.store_sales_fact.store_key = store.store_dimension.store_key
      GROUP BY store_name HAVING COUNT(*) > 70 ORDER BY store_name;

store_name | count
-----+-----
Store49    |    72
Store83    |    78
(2 rows)
```

For more detailed information, see [Joins](#). See also the Multicolumn subqueries section in [Subquery Examples](#).

Cross Joins

Also known as the Cartesian product, a cross join is the result of joining every record in one table with every record in another table. A cross join occurs when there is no join key between tables to restrict records. The following query, for example, returns all instances of vendor and store names in the vendor and store tables:

```
=> SELECT vendor_name, store_name FROM public.vendor_dimension
      CROSS JOIN store.store_dimension;

vendor_name | store_name
-----+-----
Deal Warehouse | Store41
Deal Warehouse | Store12
Deal Warehouse | Store46
Deal Warehouse | Store50
```

```
Deal Warehouse | Store15
Deal Warehouse | Store48
Deal Warehouse | Store39
Sundry Wholesale | Store41
Sundry Wholesale | Store12
Sundry Wholesale | Store46
Sundry Wholesale | Store50
Sundry Wholesale | Store15
Sundry Wholesale | Store48
Sundry Wholesale | Store39
Market Discounters | Store41
Market Discounters | Store12
Market Discounters | Store46
Market Discounters | Store50
Market Discounters | Store15
Market Discounters | Store48
Market Discounters | Store39
Market Suppliers | Store41
Market Suppliers | Store12
Market Suppliers | Store46
Market Suppliers | Store50
Market Suppliers | Store15
Market Suppliers | Store48
Market Suppliers | Store39
... | ...
(4000 rows)
```

This example's output is truncated because this particular cross join returned several thousand rows. See also [Cross Joins](#).

Subqueries

A subquery is a query nested within another query. In the following example, we want a list of all products containing the highest fat content. The inner query (subquery) returns the product containing the highest fat content among all food products to the outer query block (containing query). The outer query then uses that information to return the names of the products containing the highest fat content.

```
=> SELECT product_description, fat_content FROM public.product_dimension
      WHERE fat_content IN
          (SELECT MAX(fat_content) FROM public.product_dimension
           WHERE category_description = 'Food' AND department_description = 'Bakery')
      LIMIT 10;
      product_description | fat_content
-----|-----
Brand #59110 hotdog buns |          90
Brand #58107 english muffins |          90
Brand #57135 english muffins |          90
Brand #54870 cinnamon buns |          90
Brand #53690 english muffins |          90
Brand #53096 bagels |          90
Brand #50678 chocolate chip cookies |          90
```

```
Brand #49269 wheat bread      |      90  
Brand #47156 coffee cake    |      90  
Brand #43844 corn muffins   |      90  
(10 rows)
```

For more information, see [Subqueries](#).

Sorting Queries

Use the `ORDER BY` clause to order the rows that a query returns.

Special Note About Query Results

You could get different results running certain queries on one machine or another for the following reasons:

- [Partitioning](#) on a `FLOAT` type could return nondeterministic results because of the precision, especially when the numbers are close to one another, such as results from the `RADIANS()` function, which has a very small range of output.

To get deterministic results, use `NUMERIC` if you must partition by data that is not an `INTEGER` type.

- Most analytics (with analytic aggregations, such as `MIN()`/`MAX()`/`SUM()`/`COUNT()`/`AVG()` as exceptions) rely on a unique order of input data to get deterministic result. If the analytic [window-order](#) clause cannot resolve ties in the data, results could be different each time you run the query.

For example, in the following query, the analytic `ORDER BY` does not include the first column in the query, `promotion_key`. So for a tie of `AVG(RADIANS(cost_dollar_amount))`, `product_version`, the same `promotion_key` could have different positions within the analytic partition, resulting in a different `NTILE()` number. Thus, `DISTINCT` could also have a different result:

```
=> SELECT COUNT(*) FROM  
      (SELECT DISTINCT SIN(FLOOR(MAX(store.store_sales_fact.promotion_key))),  
       NTILE(79) OVER(PARTITION BY AVG (RADIANS  
         (store.store_sales_fact.cost_dollar_amount ))  
       ORDER BY store.store_sales_fact.product_version)  
       FROM store.store_sales_fact  
       GROUP BY store.store_sales_fact.product_version,  
                store.store_sales_fact.sales_dollar_amount ) AS store;  
count
```

```
-----  
1425  
(1 row)
```

If you add `MAX(promotion_key)` to analytic `ORDER BY`, the results are the same on any machine:

```
=> SELECT COUNT(*) FROM (SELECT DISTINCT MAX(store.store_sales_fact.promotion_key),  
    NTILE(79) OVER(PARTITION BY MAX(store.store_sales_fact.cost_dollar_amount)  
    ORDER BY store.store_sales_fact.product_version,  
    MAX(store.store_sales_fact.promotion_key))  
    FROM store.store_sales_fact  
    GROUP BY store.store_sales_fact.product_version,  
    store.store_sales_fact.sales_dollar_amount) AS store;
```

Subqueries

A subquery is a `SELECT` statement embedded within another `SELECT` statement. The embedded subquery is often referenced as the query's inner statement, while the containing query is typically referenced as the query's statement, or outer query block. A subquery returns data that the outer query uses as a condition to determine what data to retrieve.

Like any query, a subquery returns records from a table that might consist of a single column and record, a single column with multiple records, or multiple columns and records. Queries can be [noncorrelated](#) or [correlated](#). You can even use them to [update](#) or [delete](#) records in a table based on values that are stored in other database tables.

Note: Many examples in this section use the [VMart database](#).

Subqueries Used in Search Conditions

Subqueries are used as search conditions in order to filter results. They specify the conditions for the rows returned from the containing query's select-list, a query expression, or the subquery itself. The operation evaluates to TRUE, FALSE, or UNKNOWN (NULL).

Syntax

```
search-condition {
  [ { AND | OR [ NOT ] } { predicate | ( search-condition ) } ]
  } [ ,... ]
predicate
  { expression comparison-operator expression
  ... | string-expression [ NOT ] { LIKE | ILIKE | LIKEB | ILIKEB } string-expression
  ... | expression IS [ NOT ] NULL
  ... | expression [ NOT ] IN ( subquery | expression [ ,...n ] )
  ... | expression comparison-operator [ ANY | SOME ] ( subquery )
  ... | expression comparison-operator ALL ( subquery )
  ... | expression OR ( subquery )
  ... | [ NOT ] EXISTS ( subquery )
  ... | [ NOT ] IN ( subquery )
  }
```

Parameters

<p><i>search-condition</i></p>	<p>Specifies the search conditions for the rows returned from one of the following:</p> <ul style="list-style-type: none"> • Containing query's select-list • Query expression • Subquery <p>If the subquery is used with an UPDATE or DELETE statement, UPDATE specifies the rows to update and DELETE specifies the rows to delete.</p>
<p>{ AND OR NOT }</p>	<p>Keywords that specify the logical operators that combine conditions, or in the case of NOT, negate conditions.</p> <ul style="list-style-type: none"> • AND — Combines two conditions and evaluates

	<p>to TRUE when both of the conditions are TRUE.</p> <ul style="list-style-type: none"> • OR — Combines two conditions and evaluates to TRUE when either condition is TRUE. • NOT — Negates the Boolean expression specified by the predicate.
<i>predicate</i>	An expression that returns TRUE, FALSE, or UNKNOWN (NULL).
<i>expression</i>	A column name, constant, functiona scalar subquery, or combination of column names, constants, and functions connected by operators or subqueries.
<i>comparison-operator</i>	<p>Test conditions between expressions, one of the following operators:</p> <ul style="list-style-type: none"> • < tests the condition of one expression being less than the other. • > tests the condition of one expression being greater than the other. • <= tests the condition of one expression being less than or equal to the other expression. • >= tests the condition of one expression being greater than or equal to the other expression. • = tests the equality between two expressions. • <=> tests equality like the = operator, but it returns TRUE instead of UNKNOWN if both operands are UNKNOWN and FALSE instead of UNKNOWN if one operand is UNKNOWN. • <> and != test the condition of two expressions not equal to one another.
<i>string-expression</i>	A character string with optional wildcard (*) characters.
[NOT] { LIKE ILIKE LIKEB ILIKEB }	Indicates that the character string following the

	predicate is to be used (or not used) for pattern matching.
IS [NOT] NULL	Searches for values that are null or are not null.
ALL	Used with a comparison operator and a subquery. Returns TRUE for the lefthand predicate if all values returned by the subquery satisfy the comparison operation, or FALSE if not all values satisfy the comparison or if the subquery returns no rows to the outer query block.
ANY SOME	ANY and SOME are synonyms and are used with a comparison operator and a subquery. Either returns TRUE for the lefthand predicate if any value returned by the subquery satisfies the comparison operation, or FALSE if no values in the subquery satisfy the comparison or if the subquery returns no rows to the outer query block. Otherwise, the expression is UNKNOWN.
[NOT] EXISTS	Used with a subquery to test for the existence of records that the subquery returns.
[NOT] IN	Searches for an expression on the basis of an expression's exclusion or inclusion from a list. The list of values is enclosed in parentheses and can be a subquery or a set of constants.

Logical Operators AND and OR

The AND and OR logical operators combine two conditions. AND evaluates to TRUE when both of the conditions joined by the AND keyword are matched, and OR evaluates to TRUE when either condition joined by OR is matched.

OR Subqueries (complex expressions)

Vertica supports subqueries in more complex expressions using OR; for example:

- More than one subquery in the conjunct expression:

```
(SELECT MAX(b) FROM t1) + SELECT (MAX FROM t2) a IN (SELECT a FROM t1) OR b IN (SELECT x FROM t2)
```

- An OR clause in the conjunct expression involves at least one subquery:

```
a IN (SELECT a FROM t1) OR b IN (SELECT x FROM t2) a IN (SELECT a from t1) OR b = 5  
a = (SELECT MAX FROM t2) OR b = 5
```

- One subquery is present but it is part of a another expression:

```
x IN (SELECT a FROM t1) = (x = (SELECT MAX FROM t2) (x IN (SELECT a FROM t1) IS NULL
```

How AND Queries Are Evaluated

Vertica treats expressions separated by AND (conjunctive) operators individually. For example if the WHERE clause were:

```
WHERE (a IN (SELECT a FROM t1) OR b IN (SELECT x FROM t2)) AND (c IN (SELECT a FROM t1))
```

the query would be interpreted as two conjunct expressions:

1. (a IN (SELECT a FROM t1) OR b IN (SELECT x FROM t2))
2. (c IN (SELECT a FROM t1))

The first expression is considered a complex subquery, whereas the second expression is not.

Examples

The following list shows some of the ways you can filter complex conditions in the WHERE clause:

- OR expression between a subquery and a non-subquery condition:

```
=> SELECT x FROM t WHERE x > (SELECT SUM(DISTINCT x) FROM t GROUP BY y) OR x < 9;
```

- OR expression between two subqueries:

```
=> SELECT * FROM t WHERE x=(SELECT x FROM t) OR EXISTS(SELECT x FROM tt);
```

- Subquery expression:

```
=> SELECT * FROM t WHERE x=(SELECT x FROM t)+1 OR x<>(SELECT x FROM t)+1;
```

- OR expression with [NOT] IN subqueries:

```
=> SELECT * FROM t WHERE NOT (EXISTS (SELECT x FROM t)) OR x >9;
```

- OR expression with IS [NOT] NULL subqueries:

```
=> SELECT * FROM t WHERE (SELECT * FROM t)IS NULL OR (SELECT * FROM tt)IS NULL;
```

- OR expression with boolean column and subquery that returns Boolean data type:

```
=> SELECT * FROM t2 WHERE x = (SELECT x FROM t2) OR x;
```

Note: To return TRUE, the argument of OR must be a Boolean data type.

- OR expression in the CASE statement:

```
=> SELECT * FROM t WHERE CASE WHEN x=1 THEN x > (SELECT * FROM t)
      OR x < (SELECT * FROM t2) END ;
```

- Analytic function, NULL-handling function, string function, math function, and so on:

```
=> SELECT x FROM t WHERE x > (SELECT COALESCE (x,y) FROM t GROUP BY x,y) OR
      x < 9;
```

- In user-defined functions (assuming f() is one):

```
=> SELECT * FROM t WHERE x > 5 OR x = (SELECT f(x) FROM t);
```

- Use of parentheses at different places to restructure the queries:

```
=> SELECT x FROM t WHERE (x = (SELECT x FROM t) AND y = (SELECT y FROM t))
      OR (SELECT x FROM t) =1;
```

- Multicolumn subqueries:

```
=> SELECT * FROM t WHERE (x,y) = (SELECT x,y FROM t) OR x > 5;
```

- Constant/NULL on lefthand side of subquery:

```
=> SELECT * FROM t WHERE x > 5 OR 5 = (SELECT x FROM t);
```

See Also

- [Subquery Restrictions](#)

In Place of an Expression

Subqueries that return a single value (unlike a list of values returned by IN subqueries) can be used just about anywhere an expression is allowed in SQL. It can be a column name, a constant, a function, a scalar subquery, or a combination of column names, constants, and functions connected by operators or subqueries.

For example:

```
=> SELECT c1 FROM t1 WHERE c1 = ANY (SELECT c1 FROM t2) ORDER BY c1;
=> SELECT c1 FROM t1 WHERE COALESCE((t1.c1 > ANY (SELECT c1 FROM t2)), TRUE);
=> SELECT c1 FROM t1 GROUP BY c1 HAVING
    COALESCE((t1.c1 <> ALL (SELECT c1 FROM t2)), TRUE);
```

Multi-column expressions are also supported:

```
=> SELECT c1 FROM t1 WHERE (t1.c1, t1.c2) = ALL (SELECT c1, c2 FROM t2);
=> SELECT c1 FROM t1 WHERE (t1.c1, t1.c2) <> ANY (SELECT c1, c2 FROM t2);
```

Vertica returns an error on queries where more than one row would be returned by any subquery used as an expression:

```
=> SELECT c1 FROM t1 WHERE c1 = (SELECT c1 FROM t2) ORDER BY c1;
ERROR: more than one row returned by a subquery used as an expression
```

See Also

- [Subquery Restrictions](#)

Comparison Operators

Vertica supports Boolean subquery expressions in the WHERE clause with any of the following operators:

>
<
>=
<=
=
<>
<=>

WHERE clause subqueries filter results and take the following form:

```
SELECT <column, ...> FROM <table>  
WHERE <condition> (SELECT <column, ...> FROM <table> WHERE <condition>);
```

These conditions are available for all data types where comparison makes sense. All [Comparison Operators](#) are binary operators that return values of TRUE, FALSE, or UNKNOWN (NULL).

Expressions that correlate to just one outer table in the outer query block are supported, and these correlated expressions can be comparison operators.

The following subquery scenarios are supported:

```
SELECT * FROM T1 WHERE T1.x = (SELECT MAX(c1) FROM T2);  
SELECT * FROM T1 WHERE T1.x >= (SELECT MAX(c1) FROM T2 WHERE T1.y = T2.c2);  
SELECT * FROM T1 WHERE T1.x <= (SELECT MAX(c1) FROM T2 WHERE T1.y = T2.c2);
```

See Also

[Subquery Restrictions](#)

LIKE Pattern Matching

Vertica supports LIKE pattern-matching conditions in subqueries and take the following form:

string-expression [NOT] { LIKE | ILIKE | LIKEB | ILIKEB } *string-expression*

The following command searches for customers whose company name starts with "Ev" and returns the total count:

```
=> SELECT COUNT(*) FROM customer_dimension WHERE customer_name LIKE  
      (SELECT 'Ev%' FROM customer_dimension LIMIT 1);  
count  
-----  
      153  
(1 row)
```

Vertica also supports single-row subqueries as the pattern argument for LIKEB and ILIKEB predicates; for example:

```
=> SELECT * FROM t1 WHERE t1.x LIKEB (SELECT t2.x FROM t2);
```

The following symbols are substitutes for the LIKE keywords:

```
~~      LIKE  
~#      LIKEB  
~~*     ILIKE  
~#*     ILIKEB  
!~~     NOT LIKE  
!~#     NOT LIKEB  
!~~*    NOT ILIKE  
!~#*    NOT IILIKEB
```

Note: The ESCAPE keyword is not valid for the above symbols.

See [LIKE-predicate](#) in the SQL Reference Manual for additional examples.

ANY and ALL

You typically use comparison operators (=, >, <, etc.) only on subqueries that return one row. With ANY and ALL operators, you can make comparisons on subqueries that return multiple rows.

These subqueries take the following form:

```
expression comparison-operator { ANY | ALL } (subquery)
```

ANY and ALL evaluate whether any or all of the values returned by a subquery match the left-hand expression.

Equivalent Operators

You can use following operators instead of ANY or ALL:

This operator...	Is equivalent to:
SOME	ANY
IN	= ANY
NOT IN	<> ALL

Example Data

Examples below use the following tables and data:

<pre>CREATE TABLE t1 (c1 int, c2 VARCHAR(8));</pre> <pre>=> SELECT * FROM t1 ORDER BY c1; c1 c2 -----+----- 1 cab 1 abc 2 fed 2 def 3 ihg 3 ghi 4 jkl 5 mno (8 rows)</pre>	<pre>CREATE TABLE t1 (c1 int, c2 VARCHAR(8));</pre> <pre>=> SELECT * FROM t2 ORDER BY c1; c1 c2 -----+----- 1 abc 2 fed 3 jkl 3 stu 3 zzz (5 rows)</pre>
---	--

ANY Subqueries

Subqueries that use the ANY keyword return true when any value retrieved in the subquery matches the value of the left-hand expression.

Examples

An ANY subquery within an expression:

```
=> SELECT c1, c2 FROM t1 WHERE COALESCE((t1.c1 > ANY (SELECT c1 FROM t2)));
c1 | c2
-----+-----
 2 | fed
 2 | def
 3 | ihg
 3 | ghi
 4 | jkl
 5 | mno
(6 rows)
```

ANY noncorrelated subqueries without aggregates:

```
=> SELECT c1 FROM t1 WHERE c1 = ANY (SELECT c1 FROM t2) ORDER BY c1;
c1
----
 1
 1
 2
 2
 3
 3
(6 rows)
```

ANY noncorrelated subqueries with aggregates:

```
=> SELECT c1, c2 FROM t1 WHERE c1 <> ANY (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1 | c2
-----+-----
```

```
1 | cab
1 | abc
2 | fed
2 | def
4 | jkl
5 | mno
(6 rows)

=> SELECT c1 FROM t1 GROUP BY c1 HAVING c1 <> ANY (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1
----
1
2
4
5
(4 rows)
```

ANY noncorrelated subqueries with aggregates and a GROUP BY clause:

```
=> SELECT c1, c2 FROM t1 WHERE c1 <> ANY (SELECT MAX(c1) FROM t2 GROUP BY c2) ORDER BY c1;
c1 | c2
----+-----
1 | cab
1 | abc
2 | fed
2 | def
3 | ihg
3 | ghi
4 | jkl
5 | mno
(8 rows)
```

ANY noncorrelated subqueries with a GROUP BY clause:

```
=> SELECT c1, c2 FROM t1 WHERE c1 <=> ANY (SELECT c1 FROM t2 GROUP BY c1) ORDER BY c1;
c1 | c2
----+-----
1 | cab
1 | abc
2 | fed
2 | def
3 | ihg
3 | ghi
(6 rows)
```

ANY correlated subqueries with no aggregates or GROUP BY clause:

```
=> SELECT c1, c2 FROM t1 WHERE c1 >= ANY (SELECT c1 FROM t2 WHERE t2.c2 = t1.c2) ORDER BY c1;
c1 | c2
----+-----
1 | abc
2 | fed
4 | jkl
(3 rows)
```

ALL Subqueries

A subquery that uses the ALL keyword returns true when all values retrieved by the subquery match the left-hand expression, otherwise it returns false.

Examples

ALL noncorrelated subqueries without aggregates:

```
=> SELECT c1, c2 FROM t1 WHERE c1 >= ALL (SELECT c1 FROM t2) ORDER BY c1;
c1 | c2
----+-----
 3 | ihg
 3 | ghi
 4 | jkl
 5 | mno
(4 rows)
```

ALL noncorrelated subqueries with aggregates:

```
=> SELECT c1, c2 FROM t1 WHERE c1 = ALL (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1 | c2
----+-----
 3 | ihg
 3 | ghi
(2 rows)

=> SELECT c1 FROM t1 GROUP BY c1 HAVING c1 <> ALL (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1
----
 1
 2
 4
 5
(4 rows)
```

ALL noncorrelated subqueries with aggregates and a GROUP BY clause:

```
=> SELECT c1, c2 FROM t1 WHERE c1 <= ALL (SELECT MAX(c1) FROM t2 GROUP BY c2) ORDER BY c1;
c1 | c2
----+-----
 1 | cab
 1 | abc
(2 rows)
```

ALL noncorrelated subqueries with a GROUP BY clause:

```
=> SELECT c1, c2 FROM t1 WHERE c1 <> ALL (SELECT c1 FROM t2 GROUP BY c1) ORDER BY c1;
c1 | c2
----+-----
 4 | jkl
 5 | mno
(2 rows)
```

NULL Handling

Vertica supports multicolumn `<>` ALL subqueries where the columns are not marked NOT NULL. If any column contains a NULL value, Vertica returns a run-time error.

Vertica does not support `=` ANY subqueries that are nested within another expression if any column values are NULL.

See Also

[Subquery Restrictions](#)

EXISTS and NOT EXISTS

The EXISTS predicate is one of the most common predicates used to build conditions that use noncorrelated and correlated subqueries. Use EXISTS to identify the existence of a relationship without regard for the quantity. For example, EXISTS returns true if the subquery returns any rows, and [NOT] EXISTS returns true if the subquery returns no rows.

[NOT] EXISTS subqueries take the following form:

```
expression [ NOT ] EXISTS ( subquery )
```

The EXISTS condition is considered to be met if the subquery returns at least one row. Since the result depends only on whether any records are returned, and not on the contents of those records, the output list of the subquery is normally uninteresting. A common coding convention is to write all EXISTS tests as follows:

```
EXISTS (SELECT 1 WHERE ...)
```

In the above fragment, SELECT 1 returns the value 1 for every record in the query. If the query returns, for example, five records, it returns 5 ones. The system doesn't care about the real values in those records; it just wants to know if a row is returned.

Alternatively, a subquery's select list that uses EXISTS might consist of the asterisk (*). You do not need to specify column names, because the query tests for the existence or nonexistence of records that meet the conditions specified in the subquery.

```
EXISTS (SELECT * WHERE ...)
```

Notes

- If EXISTS (*subquery*) returns at least 1 row, the result is TRUE.
- If EXISTS (*subquery*) returns no rows, the result is FALSE.

- If NOT EXISTS (subquery) returns at least 1 row, the result is FALSE.
- If NOT EXISTS (subquery) returns no rows, the result is TRUE.

Examples

The following query retrieves the list of all the customers who purchased anything from any of the stores amounting to more than 550 dollars:

```
=> SELECT customer_key, customer_name, customer_state
FROM public.customer_dimension WHERE EXISTS
(SELECT 1 FROM store.store_sales_fact
WHERE customer_key = public.customer_dimension.customer_key
AND sales_dollar_amount > 550)
AND customer_state = 'MA' ORDER BY customer_key;
customer_key | customer_name | customer_state
-----+-----+-----
14818 | William X. Nielson | MA
18705 | James J. Goldberg | MA
30231 | Sarah N. McCabe | MA
48353 | Mark L. Brown | MA
(4 rows)
```

Whether you use EXISTS or IN subqueries depends on which predicates you select in outer and inner query blocks. For example, to get a list of all the orders placed by all stores on January 2, 2003 for vendors with records in the vendor table:

```
=> SELECT store_key, order_number, date_ordered
FROM store.store_orders_fact WHERE EXISTS
(SELECT 1 FROM public.vendor_dimension
WHERE public.vendor_dimension.vendor_key = store.store_orders_fact.vendor_key)
AND date_ordered = '2012-01-02';
store_key | order_number | date_ordered
-----+-----+-----
37 | 2559 | 2012-01-02
16 | 552 | 2012-01-02
35 | 1156 | 2012-01-02
13 | 3885 | 2012-01-02
25 | 554 | 2012-01-02
21 | 2687 | 2012-01-02
49 | 3251 | 2012-01-02
19 | 2922 | 2012-01-02
26 | 1329 | 2012-01-02
40 | 1183 | 2012-01-02
(10 rows)
```

The above query looks for existence of the vendor and date ordered. To return a particular value, rather than simple existence, the query looks for orders placed by the vendor who got the best deal on January 4, 2004:

```
=> SELECT store_key, order_number, date_ordered
FROM store.store_orders_fact ord, public.vendor_dimension vd
```

```
WHERE ord.vendor_key = vd.vendor_key AND vd.deal_size IN
      (SELECT MAX(deal_size) FROM public.vendor_dimension)
AND date_ordered = '2013-01-04';
store_key | order_number | date_ordered
-----+-----+-----
      166 |         36008 | 2013-01-04
      113 |         66017 | 2013-01-04
      198 |         75716 | 2013-01-04
       27 |        150241 | 2013-01-04
      148 |        182207 | 2013-01-04
        9 |       188567 | 2013-01-04
       45 |       202416 | 2013-01-04
       24 |       250295 | 2013-01-04
      121 |       251417 | 2013-01-04
(9 rows)
```

See Also

- [Subquery Restrictions](#)

IN and NOT IN

While you cannot equate a single value to a set of values, you can check to see if a single value is found within that set of values. Use the IN clause for multiple-record, single-column subqueries. After the subquery returns results introduced by IN or NOT IN, the outer query uses them to return the final result.

[NOT] IN subqueries take the following form:

```
{ expression [ NOT ] IN ( subquery ) | expression [ NOT ] IN ( expression ) }
```

There is no limit to the number of parameters passed to the IN clause of the SELECT statement; for example:

```
=> SELECT * FROM tablename WHERE column IN (a, b, c, d, e, ...);
```

Vertica also supports queries where two or more outer expressions refer to different inner expressions:

```
=> SELECT * FROM A WHERE (A.x,A.x) IN (SELECT B.x, B.y FROM B);
```

Examples

The following query uses the [VMart](#) schema to illustrate the use of outer expressions referring to different inner expressions:

```
=> SELECT product_description, product_price FROM product_dimension
      WHERE (product_dimension.product_key, product_dimension.product_key) IN
            (SELECT store.store_orders_fact.order_number,
                 store.store_orders_fact.quantity_ordered
              FROM store.store_orders_fact);
product_description | product_price
-----+-----
Brand #72 box of candy |          326
Brand #71 vanilla ice cream |          270
(2 rows)
```

To find all products supplied by stores in MA, first create the inner query and run it to ensure that it works as desired. The following query returns all stores located in MA:

```
=> SELECT store_key FROM store.store_dimension WHERE store_state = 'MA';
store_key
-----
      13
      31
(2 rows)
```

Then create the outer or main query that specifies all distinct products that were sold in stores located in MA. This statement combines the inner and outer queries using the IN predicate:

```
=> SELECT DISTINCT s.product_key, p.product_description
      FROM store.store_sales_fact s, public.product_dimension p
      WHERE s.product_key = p.product_key
            AND s.product_version = p.product_version
            AND s.store_key IN
                  (SELECT store_key
                   FROM store.store_dimension
                   WHERE store_state = 'MA')
      ORDER BY s.product_key;
product_key | product_description
-----+-----
      1 | Brand #1 white bread
      1 | Brand #4 vegetable soup
      3 | Brand #9 wheelchair
      5 | Brand #15 cheddar cheese
      5 | Brand #19 bleach
      7 | Brand #22 canned green beans
      7 | Brand #23 canned tomatoes
      8 | Brand #24 champagne
      8 | Brand #25 chicken nuggets
     11 | Brand #32 sausage
      ... | ...
(281 rows)
```

When using `NOT IN`, the subquery returns a list of zero or more values in the outer query where the comparison column does not match any of the values returned from the subquery. Using the previous example, `NOT IN` returns all the products that are not supplied from MA.

Notes

Vertica supports multicolumn NOT IN subqueries in which the columns are not marked NOT NULL. If one of the columns is found to contain a NULL value during query execution, Vertica returns a run-time error.

Similarly, IN subqueries nested within another expression are not supported if any of the column values are NULL. For example, if in the following statement column x from either table contained a NULL value, Vertica returns a run-time error:

```
=> SELECT * FROM t1 WHERE (x IN (SELECT x FROM t2)) IS FALSE;  
ERROR: NULL value found in a column used by a subquery
```

See Also

- [Subquery Restrictions](#)
- [IN-predicate](#)

Subqueries in the SELECT List

Subqueries can occur in the select list of the containing query. The results from the following statement are ordered by the first column (customer_name). You could also write ORDER BY 2 and specify that the results be ordered by the select-list subquery.

```
=> SELECT c.customer_name, (SELECT AVG(annual_income) FROM customer_dimension  
WHERE deal_size = c.deal_size) AVG_SAL_DEAL FROM customer_dimension c  
ORDER BY 1;  
customer_name | AVG_SAL_DEAL  
-----+-----  
Goldstar      | 603429  
Metatech      | 628086  
Metadata      | 666728  
Foodstar      | 695962  
Verihope      | 715683  
Veridata      | 868252  
Bettercare    | 879156  
Foodgen       | 958954  
Virtacom      | 991551  
Inicorp       | 1098835  
...
```

Notes

- Scalar subqueries in the select-list return a single row/column value. These subqueries use Boolean comparison operators: =, >, <, <>, <=, >=.

If the query is correlated, it returns NULL if the correlation results in 0 rows. If the query returns more than one row, the query errors out at run time and Vertica displays an error message that the scalar subquery must only return 1 row.

- Subquery expressions such as [NOT] IN, [NOT] EXISTS, ANY/SOME, or ALL always return a single Boolean value that evaluates to TRUE, FALSE, or UNKNOWN; the subquery itself can have many rows. Most of these queries can be correlated or noncorrelated.

Note: ALL subqueries cannot be correlated.

- Subqueries in the ORDER BY and GROUP BY clauses are supported; for example, the following statement says to order by the first column, which is the select-list subquery:

```
=> SELECT (SELECT MAX(x) FROM t2 WHERE y=t1.b) FROM t1 ORDER BY 1;
```

See Also

- [Subquery Restrictions](#)

WITH Clauses in SELECT

WITH clauses are concomitant queries within a larger, primary query. Vertica can evaluate WITH clauses in two ways:

- [Inline expansion](#) (default): Vertica evaluates each WITH clause every time it is referenced by the primary query.
- [Materialization](#): Vertica evaluates each WITH clause once, stores results in a temporary table, and references this table as often as the query requires.

For details on WITH clause syntax and requirements, see [WITH Clause](#) in the SQL Reference Manual.

Inline Expansion of WITH Clause

By default, Vertica uses inline expansion to evaluate WITH clauses. Vertica evaluates each WITH clause every time it is referenced by the primary query. Inline expansion often works best if the query does not reference the same WITH clause multiple times, or if some local optimizations are possible after inline expansion.

Example

The following example shows a WITH clause that is a good candidate for inline expansion. The WITH clause is used in a query that obtains order information for all orders shipped in 2007, between December 15-31.

```
-- Disable the materialization method if previously enabled
ALTER SESSION SET PARAMETER EnableWithClauseMaterialization=1;

-- Begin WITH clause

WITH
  store_orders_fact_new AS(
    SELECT * FROM store.store_orders_fact WHERE date_shipped between '2016-12-15' and '2016-12-31')
-- End WITH clause
-- Begin main primary query

SELECT store_key, product_key, product_version, SUM(quantity_ordered*unit_price) AS total_price
FROM store_orders_fact_new
GROUP BY store_key, product_key, product_version
ORDER BY total_price;
```

Vertica processes the query as follows:

1. Expands the WITH clause reference to `store_orders_fact_new` within the primary query.
2. After expanding the WITH clause, evaluates the primary query.

Materialization of WITH Clause

When materialization is enabled, Vertica evaluates each WITH clause once, stores results in a temporary table, and references this table as often as the query requires. Vertica drops the temporary table after primary query execution completes.

Note: If the primary query returns with an error, temporary tables might be dropped only after the client's session ends.

Materialization can facilitate better performance when `WITH` clauses are complex—for example, when the `WITH` clauses contain `JOIN` and `GROUP BY` clauses, and are referenced multiple times in the primary query.

If materialization is enabled, `WITH` statements perform an auto-commit of the user transaction. This occurs even when using `EXPLAIN` with the `WITH` statement.

Enabling Materialization

By default, materialization is disabled. You can enable and disable materialization by setting the configuration parameter `ENABLE_WITH_CLAUSE_MATERIALIZATION` at the following scopes:

- **Session:** Parameter setting remains in effect until you explicitly set or clear it, or the session ends.

```
ALTER SESSION SET PARAMETER EnableWithClauseMaterialization={ 0 | 1 };  
ALTER SESSION CLEAR PARAMETER EnableWithClauseMaterialization;
```

- **Query:** `WITH` clause includes `ENABLE_WITH_CLAUSE_MATERIALIZATION` hint. Materialization is automatically cleared when the query returns.

```
WITH /*+ENABLE_WITH_CLAUSE_MATERIALIZATION*/ with-query...
```

Example

The following example shows a `WITH` clause that is a good candidate for materialization. The query obtains data for the vendor who has the highest combined order cost for all orders:

```
-- Enable materialization  
ALTER SESSION SET PARAMETER EnableWithClauseMaterialization=1;  
  
-- Begin WITH clause, revenue  
  
WITH  
  revenue AS (  
    SELECT vendor_key, SUM(total_order_cost) AS total_revenue  
    FROM store.store_orders_fact  
    GROUP BY vendor_key ORDER BY 1)  
  
-- End defining WITH clause statement  
-- Begin main primary query  
SELECT vendor_name, vendor_address, vendor_city, total_revenue  
FROM vendor_dimension v, revenue r  
WHERE v.vendor_key = r.vendor_key AND total_revenue = (SELECT MAX(total_revenue) FROM revenue)  
ORDER BY vendor_name;
```

Vertica processes this query as follows:

1. The WITH clause revenue evaluates its SELECT statement from table store.store_orders_fact.
2. The results of the revenue clause are stored in a local temporary table.
3. Whenever the revenue clause statement is referenced, the results stored in the table are used.
4. The temporary table is dropped when query execution is complete.

Noncorrelated and Correlated Subqueries

Subqueries can be categorized into two types:

- A *noncorrelated* (simple) subquery obtains its results independently of its containing (outer) statement.
- A *correlated* subquery requires values from its outer query in order to execute.

Noncorrelated Subqueries

A noncorrelated subquery executes independently of the outer query. The subquery executes first, and then passes its results to the outer query, For example:

```
=> SELECT name, street, city, state FROM addresses WHERE state IN (SELECT state FROM states);
```

Vertica executes this query as follows:

1. Executes the subquery `SELECT state FROM states` (in bold).
2. Passes the subquery results to the outer query.

A query's WHERE and HAVING clauses can specify noncorrelated subqueries if the subquery resolves to a single row, as shown below:

In WHERE clause

```
=> SELECT COUNT(*) FROM SubQ1 WHERE SubQ1.a = (SELECT y from SubQ2);
```

In HAVING clause

```
=> SELECT COUNT(*) FROM SubQ1 GROUP BY SubQ1.a HAVING SubQ1.a = (SubQ1.a & (SELECT y from SubQ2))
```

Correlated Subqueries

A correlated subquery typically obtains values from its outer query before it executes. When the subquery returns, it passes its results to the outer query.

Note: You can use an outer join to obtain the same effect as a correlated subquery.

In the following example, the subquery needs values from the `addresses.state` column in the outer query:

```
=> SELECT name, street, city, state FROM addresses
      WHERE EXISTS (SELECT * FROM states WHERE states.state = addresses.state);
```

Vertica executes this query as follows:

1. The subquery evaluates each `addresses.state` value in the outer block records.
2. It then passes its results to the outer query block.

Flattening FROM Clause Subqueries

FROM [clause](#) subqueries are always evaluated before their containing query. In some cases, the optimizer *flattens* FROM clause subqueries so the query can execute more efficiently.

For example, in order to create a query plan for the following statement, the Vertica query optimizer evaluates all records in table `t1` before it evaluates the records in table `t0`:

```
=> SELECT * FROM (SELECT a, MAX(a) AS max FROM (SELECT * FROM t1) AS t0 GROUP BY a);
```

Given the previous query, the optimizer can internally flatten it as follows:

```
=> SELECT * FROM (SELECT a, MAX(a) FROM t1 GROUP BY a) AS t0;
```

Both queries return the same results, but the flattened query runs more quickly.

Flattening Views

When a query's FROM clause specifies a [view](#), the optimizer expands the view by replacing it with the query that the view encapsulates. If the view contains subqueries that are eligible for flattening, the optimizer produces a query plan that flattens those subqueries.

Flattening Restrictions

The optimizer cannot create a flattened query plan if a subquery or view contains one of the following elements:

- Aggregate function
- Analytic function
- Outer join (left, right or full)
- GROUP BY, ORDER BY, or HAVING clause
- DISTINCT keyword
- LIMIT or OFFSET clause
- UNION, EXCEPT, or INTERSECT clause
- EXISTS subquery

Examples

If a predicate applies to a view or subquery, the flattening operation can allow for optimizations by evaluating the predicates before the flattening takes place. Two examples follow.

View flattening

In this example, view v1 is defined as follows:

```
=> CREATE VIEW v1 AS SELECT * FROM a;
```

The following query specifies this view:

```
=> SELECT * FROM v1 JOIN b ON x=y WHERE x > 10;
```

Without flattening, the optimizer evaluates the query as follows:

1. Evaluates the subquery.
2. Applies the predicate WHERE $x > 10$.

In contrast, the optimizer can create a flattened query plan by applying the predicate before evaluating the subquery. This reduces the optimizer's work because it returns only the records WHERE $x > 10$ to the containing query.

Vertica internally transforms the previous query as follows:

```
=> SELECT * FROM (SELECT * FROM a) AS t1 JOIN b ON x=y WHERE x > 10;
```

The optimizer then flattens the query:

```
=> SELECT * FROM a JOIN b ON x=y WHERE x > 10;
```

Subquery flattening

The following example shows how Vertica transforms FROM clause subqueries within a WHERE clause IN subquery. Given the following query:

```
=> SELECT * FROM a  
WHERE b IN (SELECT b FROM (SELECT * FROM t2)) AS D WHERE x=1;
```

The optimizer flattens it as follows:

```
=> SELECT * FROM a  
WHERE b IN (SELECT b FROM t2) AS D WHERE x=1;
```

See Also

[Subquery Restrictions](#)

Subqueries in UPDATE and DELETE Statements

You can nest subqueries within [UPDATE](#) and [DELETE](#) statements.

UPDATE Subqueries

You can update records in one table according to values in others, by nesting a subquery within an UPDATE statement. The example below illustrates this through a couple of [noncorrelated subqueries](#). You can reproduce this example with the following tables:

```
CREATE TABLE addresses(cust_id INTEGER, address VARCHAR(2000));  
INSERT INTO addresses VALUES(20,'Lincoln Street');  
INSERT INTO addresses VALUES(30,'Booth Hill Road');  
INSERT INTO addresses VALUES(30,'Beach Avenue');  
INSERT INTO addresses VALUES(40,'Mt. Vernon Street');
```

```
INSERT INTO addresses VALUES(50,'Hillside Avenue');

CREATE TABLE new_addresses(new_cust_id integer, new_address Boolean DEFAULT 'T');
INSERT INTO new_addresses VALUES(20);
INSERT INTO new_addresses VALUES(30);
INSERT INTO new_addresses VALUES(60,'F');
INSERT INTO new_addresses VALUES(80,'T');
COMMIT;
```

Queries on these tables return the following results:

```
=> SELECT * FROM addresses;
cust_id |      address
-----+-----
      50 | Hillside Avenue
      30 | Booth Hill Road
      40 | Mt. Vernon Street
      20 | Lincoln Street
      30 | Beach Avenue
(5 rows)

=> SELECT * FROM new_addresses;
new_cust_id | new_address
-----+-----
          30 | t
          20 | t
          80 | t
          60 | f
(4 rows)
```

1. The following UPDATE statement uses a [noncorrelated subquery](#) to join `new_addresses` and `addresses` records on customer IDs. UPDATE sets the value 'New Address' in the joined `addresses` records. The statement output indicates that three rows were updated:

```
=> UPDATE addresses SET address='New Address'
   WHERE cust_id IN (SELECT new_cust_id FROM new_addresses WHERE new_address='T');
OUTPUT
-----
3
(1 row)
```

2. Query the `addresses` table to see the changes for matching customer ID 20 and 30. Addresses for customer ID 40 and 50 are not updated:

```
=> SELECT * FROM addresses;
cust_id |      address
-----+-----
      20 | New Address
      30 | New Address
      30 | New Address
      40 | Mt. Vernon Street
      50 | Hillside Avenue
(5 rows)
```

```
=>COMMIT;  
COMMIT
```

DELETE Subqueries

You can delete records in one table based according to values in others by nesting a subquery within a **DELETE** statement.

For example, you want to remove records from `new_addresses` that were used earlier to update records in `addresses`. The following **DELETE** statement uses a **noncorrelated subquery** to join `new_addresses` and `addresses` records on customer IDs. It then deletes the joined records from table `new_addresses`:

```
=> DELETE FROM new_addresses  
      WHERE new_cust_id IN (SELECT cust_id FROM addresses WHERE address='New Address');  
OUTPUT  
-----  
      2  
(1 row)  
  
=> COMMIT;  
COMMIT
```

Querying `new_addresses` confirms that the records were deleted:

```
=> SELECT * FROM new_addresses;  
new_cust_id | new_address  
-----+-----  
          60 | f  
          80 | t  
(2 rows)
```

Subquery Examples

This topic illustrates some of the subqueries you can write. The examples use the **VMart** example database.

Single-Row Subqueries

Single-row subqueries are used with single-row comparison operators (`=`, `>=`, `<=`, `<>`, and `<=>`) and return exactly one row.

For example, the following query retrieves the name and hire date of the oldest employee in the Vmart database:

```
=> SELECT employee_key, employee_first_name, employee_last_name, hire_date
      FROM employee_dimension
      WHERE hire_date = (SELECT MIN(hire_date) FROM employee_dimension);
employee_key | employee_first_name | employee_last_name | hire_date
-----+-----+-----+-----
          2292 | Mary                | Bauer              | 1956-01-11
(1 row)
```

Multiple-Row Subqueries

Multiple-row subqueries return multiple records.

For example, the following IN clause subquery returns the names of the employees making the highest salary in each of the six regions:

```
=> SELECT employee_first_name, employee_last_name, annual_salary, employee_region
      FROM employee_dimension WHERE annual_salary IN
      (SELECT MAX(annual_salary) FROM employee_dimension GROUP BY employee_region)
      ORDER BY annual_salary DESC;
employee_first_name | employee_last_name | annual_salary | employee_region
-----+-----+-----+-----
Alexandra           | Sanchez            | 992363        | West
Mark                 | Vogel              | 983634        | South
Tiffany             | Vu                 | 977716        | SouthWest
Barbara             | Lewis              | 957949        | MidWest
Sally               | Gauthier           | 927335        | East
Wendy               | Nielson            | 777037        | NorthWest
(6 rows)
```

Multicolumn Subqueries

Multicolumn subqueries return one or more columns. Sometimes a subquery's result set is evaluated in the containing query in column-to-column and row-to-row comparisons.

Note: Multicolumn subqueries can use the <>, !=, and = operators but not the <, >, <=, >= operators.

You can substitute some multicolumn subqueries with a join, with the reverse being true as well. For example, the following two queries ask for the sales transactions of all products sold online to customers located in Massachusetts and return the same result set. The only difference is the first query is written as a join and the second is written as a subquery.

Join query:	Subquery:
<pre>=> SELECT * FROM online_sales.online_sales_fact INNER JOIN public.customer_dimension USING (customer_key)</pre>	<pre>=> SELECT * FROM online_sales.online_sales_fact WHERE customer_key IN (SELECT customer_key</pre>

WHERE customer_state = 'MA';	FROM public.customer_dimension WHERE customer_state = 'MA');
------------------------------	---

The following query returns all employees in each region whose salary is above the average:

```
=> SELECT e.employee_first_name, e.employee_last_name, e.annual_salary,
       e.employee_region, s.average
FROM employee_dimension e,
     (SELECT employee_region, AVG(annual_salary) AS average
      FROM employee_dimension GROUP BY employee_region) AS s
WHERE e.employee_region = s.employee_region AND e.annual_salary > s.average
ORDER BY annual_salary DESC;
```

employee_first_name	employee_last_name	annual_salary	employee_region	average
Doug	Overstreet	995533	East	61192.786013986
Matt	Gauthier	988807	South	57337.8638902996
Lauren	Nguyen	968625	West	56848.4274914089
Jack	Campbell	963914	West	56848.4274914089
William	Martin	943477	NorthWest	58928.2276119403
Luigi	Campbell	939255	MidWest	59614.9170454545
Sarah	Brown	901619	South	57337.8638902996
Craig	Goldberg	895836	East	61192.786013986
Sam	Vu	889841	MidWest	59614.9170454545
Luigi	Sanchez	885078	MidWest	59614.9170454545
Michael	Weaver	882685	South	57337.8638902996
Doug	Pavlov	881443	SouthWest	57187.2510548523
Ruth	McNulty	874897	East	61192.786013986
Luigi	Dobisz	868213	West	56848.4274914089
Laura	Lang	865829	East	61192.786013986
...				

You can also use the **EXCEPT**, **INTERSECT**, and **UNION [ALL]** keywords in FROM, WHERE, and HAVING clauses.

The following subquery returns information about all Connecticut-based customers who bought items through either stores or online sales channel and whose purchases amounted to more than 500 dollars:

```
=> SELECT DISTINCT customer_key, customer_name FROM public.customer_dimension
       WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact
                             WHERE sales_dollar_amount > 500
                             UNION ALL
                             SELECT customer_key FROM online_sales.online_sales_fact
                             WHERE sales_dollar_amount > 500)
       AND customer_state = 'CT';
```

customer_key	customer_name
200	Carla Y. Kramer
733	Mary Z. Vogel
931	Lauren X. Roy
1533	James C. Vu
2948	Infocare
4909	Matt Z. Winkler
5311	John Z. Goldberg
5520	Laura M. Martin
5623	Daniel R. Kramer

HAVING Clause Subqueries

A HAVING clause is used in conjunction with the GROUP BY clause to filter the select-list records that a GROUP BY returns. HAVING clause subqueries must use Boolean comparison operators: =, >, <, <>, <=, >= and take the following form:

```
SELECT <column, ...>
FROM <table>
GROUP BY <expression>
HAVING <expression>
  (SELECT <column, ...>
   FROM <table>
   HAVING <expression>);
```

For example, the following statement uses the [VMart](#) database and returns the number of customers who purchased lowfat products. Note that the GROUP BY clause is required because the query uses an aggregate (COUNT).

```
=> SELECT s.product_key, COUNT(s.customer_key) FROM store.store_sales_fact s
   GROUP BY s.product_key HAVING s.product_key IN
   (SELECT product_key FROM product_dimension WHERE diet_type = 'Low Fat');
```

The subquery first returns the product keys for all low-fat products, and the outer query then counts the total number of customers who purchased those products.

```
product_key | count
-----+-----
         15 |      2
         41 |      1
         66 |      1
        106 |      1
        118 |      1
        169 |      1
        181 |      2
        184 |      2
        186 |      2
        211 |      1
        229 |      1
        267 |      1
        289 |      1
        334 |      2
        336 |      1
(15 rows)
```

Subquery Restrictions

The following list summarizes subquery restrictions in Vertica.

- Subqueries are not allowed in the defining query of a **CREATE PROJECTION** statement.
- Subqueries can be used in the **SELECT** list, but **GROUP BY** or aggregate functions are not allowed in the query if the subquery is not part of the **GROUP BY** clause in the containing query. For example, the following two statements return an error message:

```
=> SELECT y, (SELECT MAX(a) FROM t1) FROM t2 GROUP BY y;  
ERROR: subqueries in the SELECT or ORDER BY are not supported if the  
subquery is not part of the GROUP BY  
=> SELECT MAX(y), (SELECT MAX(a) FROM t1) FROM t2;  
ERROR: subqueries in the SELECT or ORDER BY are not supported if the  
query has aggregates and the subquery is not part of the GROUP BY
```

- Subqueries are supported within **UPDATE** statements with the following exceptions:
 - You cannot use **SET column = {expression}** to specify a subquery.
 - The table specified in the **UPDATE** list cannot also appear in the **FROM** list (no self joins).
- **FROM** clause subqueries require an alias but tables do not. If the table has no alias, the query must refer to columns inside it as *table-name.column-name*. However, if column names are uniquely identified among all tables used by the query, then column names do not need to include their table name.
- If the **ORDER BY** clause is inside a **FROM** clause subquery, rather than in the containing query, the query could return unexpected sort results. This is because Vertica data comes from multiple nodes, so sort order cannot be guaranteed unless an **ORDER BY** clause is specified in the outer query block. This behavior is compliant with the SQL standard but it might differ from other databases.
- Multicolumn subqueries cannot use the **<**, **>**, **<=**, **>=** comparison operators. They can use **<>**, **!=**, and **=** operators.
- **WHERE** and **HAVING** clause subqueries must use Boolean comparison operators: **=**, **>**, **<**, **<>**, **<=**, **>=**. Those subqueries can be noncorrelated and correlated.
- **[NOT] IN** and **ANY** subqueries nested within another expression are not supported if any of the column values are **NULL**. In the following statement, for example, if column **x** from either table **t1** or **t2** contains a **NULL** value, Vertica returns a run-time error:

```
=> SELECT * FROM t1 WHERE (x IN (SELECT x FROM t2)) IS FALSE;  
ERROR: NULL value found in a column used by a subquery
```

- Vertica returns an error message during subquery run time on scalar subqueries that return more than one row.
- Aggregates and GROUP BY clauses are allowed in subqueries, as long as those subqueries are not correlated.
- Correlated expressions under ALL and [NOT] IN are not supported.
- Correlated expressions under OR are not supported.
- Multiple correlations are allowed only for subqueries that are joined with an equality predicate (<, >, <=, >=, =, <>, <=>) but IN/NOT IN, EXISTS/NOT EXISTS predicates within correlated subqueries are not allowed:

```
=> SELECT t2.x, t2.y, t2.z FROM t2 WHERE t2.z NOT IN  
      (SELECT t1.z FROM t1 WHERE t1.x = t2.x);  
ERROR: Correlated subquery with NOT IN is not supported
```

- Up to one level of correlated subqueries is allowed in the WHERE clause if the subquery references columns in the immediate outer query block. For example, the following query is not supported because the `t2.x = t3.x` subquery can only refer to table `t1` in the outer query, making it a correlated expression because `t3.x` is two levels out:

```
=> SELECT t3.x, t3.y, t3.z FROM t3 WHERE t3.z IN (  
      SELECT t1.z FROM t1 WHERE EXISTS (  
        SELECT 'x' FROM t2 WHERE t2.x = t3.x) AND t1.x = t3.x);  
ERROR: More than one level correlated subqueries are not supported
```

The query is supported if it is rewritten as follows:

```
=> SELECT t3.x, t3.y, t3.z FROM t3 WHERE t3.z IN  
      (SELECT t1.z FROM t1 WHERE EXISTS  
        (SELECT 'x' FROM t2 WHERE t2.x = t1.x)  
        AND t1.x = t3.x);
```

Joins

Queries can combine records from multiple tables, or multiple instances of the same table. A query that combines records from one or more tables is called a join. Joins are allowed in SELECT statements and subqueries.

Supported Join Types

Vertica supports the following join types:

- Inner (including natural, cross) joins
- Left, right, and full outer joins
- Optimizations for equality and range joins predicates

Vertica does not support nested loop joins.

Join Algorithms

Vertica's query optimizer implements joins with either the hash join or merge join algorithm. For details, see [Hash Joins Versus Merge Joins](#).

Join Syntax

Vertica supports the ANSI SQL-92 standard for joining tables, as follows:

```
table-reference [join-type] JOIN table-reference [ ON join-predicate ]
```

where *join-type* can be one of the following:

- **INNER** (default)
- **LEFT [OUTER]**
- **RIGHT [OUTER]**
- **FULL [OUTER]**

- [NATURAL](#)
- [CROSS](#)

For example:

```
=> SELECT * FROM T1 INNER JOIN T2 ON T1.id = T2.id;
```

Note: The ON *join-predicate* clause is invalid for NATURAL and CROSS joins, required for all other join types.

Alternative Syntax Options

Vertica also supports two older join syntax conventions:

Join specified by WHERE clause join predicate

INNER JOIN is equivalent to a query that specifies its join predicate in a WHERE clause. For example, this example and the previous one return equivalent results. They both specify an inner join between tables T1 and T2 on columns T1.id and T2.id, respectively.

```
=> SELECT * FROM T1, T2 WHERE T1.id = T2.id;
```

Join specified by USING clause

You can join two tables on identically named columns in a USING clause. For example:

```
=> SELECT * FROM T1 INNER JOIN T2 USING(id);
```

The INNER keyword is optional; a join that is specified by a USING clause is always an inner join. Thus, this example and the previous ones return equivalent results.

Benefits of SQL-92 Join Syntax

Vertica recommends that you use SQL-92 join syntax for several reasons:

- SQL-92 outer join syntax is portable across databases; the older syntax was not consistent between databases.
- SQL-92 syntax provides greater control over whether predicates are evaluated during or after outer joins. This was also not consistent between databases when using the older syntax.

- SQL-92 syntax eliminates ambiguity in the order of evaluating the joins, in cases where more than two tables are joined with outer joins.

Join Conditions vs. Filter Conditions

If you do not use the SQL-92 syntax, join conditions (predicates that are evaluated during the join) are difficult to distinguish from filter conditions (predicates that are evaluated after the join), and in some cases cannot be expressed at all. With SQL-92, join conditions and filter conditions are separated into two different clauses, the `ON` clause and the `WHERE` clause, respectively, making queries easier to understand.

- **The `ON` clause** contains relational operators (for example, `<`, `<=`, `>`, `>=`, `<>`, `=`, `<=>`) or other predicates that specify which records from the left and right input relations to combine, such as by matching foreign keys to primary keys. `ON` can be used with inner, left outer, right outer, and full outer joins. Cross joins and union joins do not use an `ON` clause.

Inner joins return all pairings of rows from the left and right relations for which the `ON` clause evaluates to `TRUE`. In a left join, all rows from the left relation in the join are present in the result; any row of the left relation that does not match any rows in the right relation is still present in the result but with nulls in any columns taken from the right relation. Similarly, a right join preserves all rows from the right relation, and a full join retains all rows from both relations.

- **The `WHERE` clause** is evaluated after the join is performed. It filters records returned by the `FROM` clause, eliminating any records that do not satisfy the `WHERE` clause condition.

Vertica automatically converts outer joins to inner joins in cases where it is correct to do so, allowing the optimizer to choose among a wider set of query plans and leading to better performance.

Inner Joins

An inner join combines records from two tables based on a join predicate and requires that each record in the first table has a matching record in the second table. Thus, inner joins return only records from both joined tables that satisfy the join condition. Records that contain no matches are excluded from the result set.

Inner joins take the following form:

```
SELECT column-list FROM left-join-table  
[INNER] JOIN right-join-table ON join-predicate
```

If you omit the INNER keyword, Vertica assumes an inner join. Inner joins are commutative and associative. You can specify tables in any order without changing the results.

Example

The following example specifies an inner join between tables `store.store_dimension` and `public.employee_dimension` whose records have matching values in columns `store_region` and `employee_region`, respectively:

```
=> SELECT s.store_region, SUM(e.vacation_days) TotalVacationDays  
FROM public.employee_dimension e  
JOIN store.store_dimension s ON s.store_region=e.employee_region  
GROUP BY s.store_region ORDER BY TotalVacationDays;
```

This join can also be expressed as follows:

```
=> SELECT s.store_region, SUM(e.vacation_days) TotalVacationDays  
FROM public.employee_dimension e, store.store_dimension s  
WHERE s.store_region=e.employee_region  
GROUP BY s.store_region ORDER BY TotalVacationDays;
```

Both queries return the same result set:

```
store_region | TotalVacationDays  
-----+-----  
NorthWest   |          23280  
SouthWest   |          367250  
MidWest     |          925938  
South       |         1280468  
East        |         1952854  
West        |         2849976  
(6 rows)
```

If the join's inner table `store.store_dimension` has any rows with `store_region` values that do not match `employee_region` values in table `public.employee_dimension`, those rows are excluded from the result set. To include that row, you can specify an [outer join](#).

Equi-Joins and Non Equi-Joins

Vertica supports any arbitrary join expression with both matching and non-matching column values. For example:

```
SELECT * FROM fact JOIN dim ON fact.x = dim.x;
SELECT * FROM fact JOIN dim ON fact.x > dim.y;
SELECT * FROM fact JOIN dim ON fact.x <= dim.y;
SELECT * FROM fact JOIN dim ON fact.x <> dim.y;
SELECT * FROM fact JOIN dim ON fact.x <=> dim.y;
```

Note: Operators `=` and `<=>` generally run the fastest.

Equi-joins are based on equality (matching column values). This equality is indicated with an equal sign (`=`), which functions as the comparison operator in the `ON` clause using SQL-92 syntax or the `WHERE` clause using older join syntax.

The first example below uses SQL-92 syntax and the `ON` clause to join the online sales table with the call center table using the call center key; the query then returns the sale date key that equals the value 156:

```
=> SELECT sale_date_key, cc_open_date FROM online_sales.online_sales_fact
      INNER JOIN   online_sales.call_center_dimension
      ON (online_sales.online_sales_fact.call_center_key =
          online_sales.call_center_dimension.call_center_key
          AND sale_date_key = 156);
sale_date_key | cc_open_date
-----+-----
          156 | 2005-08-12
(1 row)
```

The second example uses older join syntax and the `WHERE` clause to join the same tables to get the same results:

```
=> SELECT sale_date_key, cc_open_date
      FROM online_sales.online_sales_fact, online_sales.call_center_dimension
      WHERE online_sales.online_sales_fact.call_center_key =
            online_sales.call_center_dimension.call_center_key
            AND sale_date_key = 156;
sale_date_key | cc_open_date
-----+-----
          156 | 2005-08-12
(1 row)
```

Vertica also permits tables with compound (multiple-column) primary and foreign keys. For example, to create a pair of tables with multi-column keys:

```
=> CREATE TABLE dimension(pk1 INTEGER NOT NULL, pk2 INTEGER NOT NULL);=> ALTER TABLE dimension ADD  
PRIMARY KEY (pk1, pk2);  
=> CREATE TABLE fact (fk1 INTEGER NOT NULL, fk2 INTEGER NOT NULL);  
=> ALTER TABLE fact ADD FOREIGN KEY (fk1, fk2) REFERENCES dimension (pk1, pk2);
```

To join tables using compound keys, you must connect two [join predicates](#) with a Boolean AND operator. For example:

```
=> SELECT * FROM fact f JOIN dimension d ON f.fk1 = d.pk1 AND f.fk2 = d.pk2;
```

You can write queries with expressions that contain the `<=>` operator for NULL=NULL joins.

```
=> SELECT * FROM fact JOIN dim ON fact.x <=> dim.y;
```

The `<=>` operator performs an equality comparison like the `=` operator, but it returns true, instead of NULL, if both operands are NULL, and false, instead of NULL, if one operand is NULL.

```
=> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;  
?column? | ?column? | ?column?  
-----+-----+-----  
t        | t        | f  
(1 row)
```

Compare the `<=>` operator to the `=` operator:

```
=> SELECT 1 = 1, NULL = NULL, 1 = NULL;  
?column? | ?column? | ?column?  
-----+-----+-----  
t        |          |  
(1 row)
```

Note: Writing NULL=NULL joins on primary key/foreign key combinations is not an optimal choice because PK/FK columns are usually defined as NOT NULL.

When composing joins, it helps to know in advance which columns contain null values. An employee's hire date, for example, would not be a good choice because it is unlikely hire date would be omitted. An hourly rate column, however, might work if some employees are paid hourly and some are salaried. If you are unsure about the value of columns in a given table and want to check, type the command:

```
=> SELECT COUNT(*) FROM tablename WHERE columnname IS NULL;
```

Natural Joins

A natural join is just a join with an implicit join predicate. Natural joins can be inner, left outer, right outer, or full outer joins and take the following form:

```
SELECT column-list FROM left-join-table  
NATURAL [ INNER | LEFT OUTER | RIGHT OUTER | FULL OUTER ] JOIN right-join-table ON join-predicate
```

Natural joins are, by default, natural *inner* joins; however, there can also be natural (left/right) outer joins. The primary difference between an inner and natural join is that inner joins have an explicit join condition, whereas the natural join's conditions are formed by matching all pairs of columns in the tables that have the same name and compatible data types, making natural joins equi-joins because join condition are equal between common columns. (If the data types are incompatible, Vertica returns an error.)

Note: The [Data Type Coercion Chart](#) lists the data types that can be cast to other data types. If one data type can be cast to the other, those two data types are compatible.

The following query is a simple natural join between tables T1 and T2 when the T2 column val is greater than 5:

```
=> SELECT * FROM T1 NATURAL JOIN T2 WHERE T2.val > 5;
```

The following example shows a natural join between the `store_sales_fact` table and the `product_dimension` table with columns of the same name, `product_key` and `product_version`:

```
=> SELECT product_description, store.store_sales_fact.*  
FROM store.store_sales_fact, public.product_dimension  
WHERE store.store_sales_fact.product_key = public.product_dimension.product_key  
AND store.store_sales_fact.product_version = public.product_dimension.product_version;
```

The following three queries return the same result expressed as a basic query, an inner join, and a natural join. at the table expressions are equivalent only if the common attribute in the `store_sales_fact` table and the `store_dimension` table is `store_key`. If both tables have a column named `store_key`, then the natural join would also have a `store_sales_fact.store_key = store_dimension.store_key` join condition. Since the results are the same in all three instances, they are shown in the first (basic) query only:

```
=> SELECT store_name FROM store.store_sales_fact, store.store_dimension  
WHERE store.store_sales_fact.store_key = store.store_dimension.store_key  
AND store.store_dimension.store_state = 'MA' ORDER BY store_name;  
store_name  
-----  
Store11  
Store128  
Store178  
Store66  
Store8  
Store90  
(6 rows)
```

The query written as an inner join:

```
=> SELECT store_name FROM store.store_sales_fact
      INNER JOIN store.store_dimension
      ON (store.store_sales_fact.store_key = store.store_dimension.store_key)
      WHERE store.store_dimension.store_state = 'MA' ORDER BY store_name;
```

In the case of the natural join, the join predicate appears implicitly by comparing all of the columns in both tables that are joined by the same column name. The result set contains only one column representing the pair of equally-named columns.

```
=> SELECT store_name FROM store.store_sales_fact
      NATURAL JOIN store.store_dimension
      WHERE store.store_dimension.store_state = 'MA' ORDER BY store_name;
```

Cross Joins

Cross joins are the simplest joins to write, but they are not usually the fastest to run because they consist of all possible combinations of two tables' records. Cross joins contain no join condition and return what is known as a Cartesian product, where the number of rows in the result set is equal to the number of rows in the first table multiplied by the number of rows in the second table.

The following query returns all possible combinations from the promotion table and the store sales table:

```
=> SELECT * FROM promotion_dimension CROSS JOIN store.store_sales_fact;
```

Because this example returns over 600 million records, many cross join results can be extremely large and difficult to manage. Cross joins can be useful, however, such as when you want to return a single-row result set.

Tip: Filter out unwanted records in a cross with `WHERE` clause join predicates:

```
=> SELECT * FROM promotion_dimension p CROSS JOIN store.store_sales_fact f
      WHERE p.promotion_key LIKE f.promotion_key;
```

Implicit versus Explicit Joins

Vertica recommends that you do not write implicit cross joins (comma-separated tables in the `FROM` clause). These queries can imply accidental omission of a join predicate.

The following query implicitly cross joins tables `promotion_dimension` and `store.store_sales_fact`:

```
=> SELECT * FROM promotion_dimension, store.store_sales_fact;
```

It is better practice to express this cross join explicitly, as follows:

```
=> SELECT * FROM promotion_dimension CROSS JOIN store.store_sales_fact;
```

Examples

The following example creates two small tables and their superprojections and then runs a cross join on the tables:

```
=> CREATE TABLE employee(employee_id INT, employee_fname VARCHAR(50));
=> CREATE TABLE department(dept_id INT, dept_name VARCHAR(50));
=> INSERT INTO employee VALUES (1, 'Andrew');
=> INSERT INTO employee VALUES (2, 'Priya');
=> INSERT INTO employee VALUES (3, 'Michelle');
=> INSERT INTO department VALUES (1, 'Engineering');
=> INSERT INTO department VALUES (2, 'QA');
=> SELECT * FROM employee CROSS JOIN department;
```

In the result set, the cross join retrieves records from the first table and then creates a new row for every row in the 2nd table. It then does the same for the next record in the first table, and so on.

```
employee_id | employee_name | dept_id | dept_name
-----+-----+-----+-----
          1 | Andrew       |        1 | Engineering
          2 | Priya        |        1 | Engineering
          3 | Michelle     |        1 | Engineering
          1 | Andrew       |        2 | QA
          2 | Priya        |        2 | QA
          3 | Michelle     |        2 | QA
(6 rows)
```

Outer Joins

Outer joins extend the functionality of inner joins by letting you preserve rows of one or both tables that do not have matching rows in the non-preserved table. Outer joins take the following form:

```
SELECT column-list FROM left-join-table
[ LEFT | RIGHT | FULL ] OUTER JOIN right-join-table ON join-predicate
```

Note: Omitting the keyword OUTER from your statements does not affect results of left and right joins. LEFT OUTER JOIN and LEFT JOIN perform the same operation and return the same results.

Left Outer Joins

A left outer join returns a complete set of records from the left-joined (preserved) table T1, with matched records, where available, in the right-joined (non-preserved) table T2. Where Vertica finds no match, it extends the right side column (T2) with null values.

```
=> SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.x = T2.x;
```

To exclude the non-matched values from T2, write the same left outer join, but filter out the records you don't want from the right side by using a `WHERE` clause:

```
=> SELECT * FROM T1 LEFT OUTER JOIN T2  
ON T1.x = T2.x WHERE T2.x IS NOT NULL;
```

The following example uses a left outer join to enrich telephone call detail records with an incomplete numbers dimension. It then filters out results that are known not to be from Massachusetts:

```
=> SELECT COUNT(*) FROM calls LEFT OUTER JOIN numbers  
ON calls.to_phone = numbers.phone WHERE NVL(numbers.state, '') <> 'MA';
```

Right Outer Joins

A right outer join returns a complete set of records from the right-joined (preserved) table, as well as matched values from the left-joined (non-preserved) table. If Vertica finds no matching records from the left-joined table (T1), `NULL` values appears in the T1 column for any records with no matching values in T1. A right join is, therefore, similar to a left join, except that the treatment of the tables is reversed.

```
=> SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.x = T2.x;
```

The above query is equivalent to the following query, where `T1 RIGHT OUTER JOIN T2 = T2 LEFT OUTER JOIN T1`.

```
=> SELECT * FROM T2 LEFT OUTER JOIN T1 ON T2.x = T1.x;
```

The following example identifies customers who have *not* placed an order:

```
=> SELECT customers.customer_id FROM orders RIGHT OUTER JOIN customers  
ON orders.customer_id = customers.customer_id  
GROUP BY customers.customer_id HAVING COUNT(orders.customer_id) = 0;
```

Full Outer Joins

A full outer join returns results for both left and right outer joins. The joined table contains all records from both tables, including nulls (missing matches) from either side of the join. This is useful if you want to see, for example, each employee who is assigned to a particular department and each department that has an employee, but you also want to see all the employees who are not assigned to a particular department, as well as any department that has no employees:

```
=> SELECT employee_last_name, hire_date FROM employee_dimension emp  
FULL OUTER JOIN department dept ON emp.employee_key = dept.department_key;
```

Notes

Vertica also supports joins where the outer (preserved) table or subquery is replicated on more than one node and the inner (non-preserved) table or subquery is segmented across more than one node. For example, in the following query, the fact table, which is almost always segmented, appears on the non-preserved side of the join, and it is allowed:

```
=> SELECT sales_dollar_amount, transaction_type, customer_name  
FROM store.store_sales_fact f RIGHT JOIN customer_dimension d  
ON f.customer_key = d.customer_key;
```

sales_dollar_amount	transaction_type	customer_name
252	purchase	Inistar
363	purchase	Inistar
510	purchase	Inistar
-276	return	Foodcorp
252	purchase	Foodcorp
195	purchase	Foodcorp
290	purchase	Foodcorp
222	purchase	Foodcorp
		Foodgen
		Goldcare

(10 rows)

Controlling Join Inputs

By default, the optimizer uses its own internal logic to determine whether to join one table to another as an inner or outer input. Occasionally, the optimizer might choose the larger table as the inner input to a join. Doing so can incur performance and concurrency issues.

If the configuration parameter `EnableForceOuter` is set to 1, you can control join inputs for specific tables through `ALTER TABLE . . FORCE OUTER`. The

`FORCE OUTER` option modifies a table's `force_outer` setting in the system table [TABLES](#). When implementing a join, Vertica compares the `force_outer` settings of the participating tables:

- If table settings are unequal, Vertica uses them to set the join inputs:
 - A table with a low `force_outer` setting relative to other tables is joined to them as an inner input.
 - A table with a high `force_outer` setting relative to other tables is joined to them as an outer input.
- If all table settings are equal, Vertica ignores them and assembles the join on its own.

The `force_outer` column is initially set to 5 for all newly-defined tables. You can use [ALTER TABLE . . FORCE OUTER](#) to reset `force_outer` to a value equal to or greater than 0. For example, you might change the `force_outer` settings of tables `abc` and `xyz` to 3 and 8, respectively:

```
=> ALTER TABLE abc FORCE OUTER 3;  
=> ALTER TABLE xyz FORCE OUTER 8;
```

Given these settings, the optimizer joins `abc` as the inner input to any table with a `force_outer` value greater than 3. The optimizer joins `xyz` as the outer input to any table with a `force_outer` value less than 8.

Projection Inheritance

When you query a projection directly, it inherits the `force_outer` setting of its anchor table. The query then uses this setting when joined to another projection.

Enabling Forced Join Inputs

The configuration parameter `EnableForceOuter` determines whether Vertica uses a table's `force_outer` value to implement a join. By default, this parameter is set to 0, and forced join inputs are disabled. You can enable forced join inputs at session and database scopes, through [ALTER SESSION](#) and [ALTER DATABASE](#), respectively:

```
=> ALTER SESSION SET EnableForceOuter = { 0 | 1 };  
=> ALTER DATABASE db-name SET EnableForceOuter = { 0 | 1 };
```

If `EnableForceOuter` is set to 0, `ALTER TABLE . . FORCE OUTER` statements return with this warning:

```
WARNING 0: Set configuration parameter EnableForceOuter for the current session or the database in order to use force_outer value
```

Viewing Forced Join Inputs

EXPLAIN-generated query plans indicate whether the configuration parameter `EnableForceOuter` is on. A join query might include tables whose `force_outer` settings are less or greater than the default value of 5. In this case, the query plan includes a `Force outer level` field for the relevant join inputs.

For example, the following query joins tables `store.store_sales` and `public.products`, where both tables have the same `force_outer` setting (5). `EnableForceOuter` is on, as indicated in the generated query plan:

```
=> EXPLAIN SELECT s.store_key, p.product_description, s.sales_quantity, s.sale_date
FROM store.store_sales s JOIN public.products p ON s.product_key=p.product_key
WHERE s.sale_date='2014-12-01' ORDER BY s.store_key, s.sale_date;

EnableForceOuter is on
Access Path:
+-SORT [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +---> JOIN HASH [Cost: 5K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Outer (BROADCAST)(LOCAL ROUND ROBIN)
| | Join Cond: (sales.product_key = products.product_key)
| | Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 3)
| | | Projection: store.store_sales_b0
| | | Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | Filter: (sales.sale_date = '2014-12-01'::date)
| | | Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for products [Cost: 177, Rows: 60K (NO STATISTICS)] (PATH ID: 4)
| | | Projection: public.products_b0
| | | Materialize: products.product_key, products.product_description
| | | Execute on: All Nodes
```

The following `ALTER TABLE` statement resets the `force_outer` setting of `public.products` to 1:

```
=> ALTER TABLE public.products FORCE OUTER 1;
ALTER TABLE
```

The regenerated query plan for the same join now includes a `Force outer level` field and specifies `public.products` as the inner input:

```
=> EXPLAIN SELECT s.store_key, p.product_description, s.sales_quantity, s.sale_date
FROM store.store_sales s JOIN public.products p ON s.product_key=p.product_key
WHERE s.sale_date='2014-12-01' ORDER BY s.store_key, s.sale_date;
```

```
EnableForceOuter is on
Access Path:
+-SORT [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +--> JOIN HASH [Cost: 5K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Outer (BROADCAST)(LOCAL ROUND
ROBIN)
| | Join Cond: (sales.product_key = products.product_key)
| | Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 3)
| | | Projection: store.store_sales_b0
| | | Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | Filter: (sales.sale_date = '2014-12-01'::date)
| | | Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for products [Cost: 177, Rows: 60K (NO STATISTICS)] (PATH ID: 4)
| | | Projection: public.products_b0
| | | Force outer level: 1
| | | Materialize: products.product_key, products.product_description
| | | Execute on: All Nodes
```

If you change the `force_outer` setting of `public.products` to 8, Vertica creates a different query plan that specifies `public.products` as the outer input:

```
=> ALTER TABLE public.products FORCE OUTER 8;
ALTER TABLE

=> EXPLAIN SELECT s.store_key, p.product_description, s.sales_quantity, s.sale_date
FROM store.store_sales s JOIN public.products p ON s.product_key=p.product_key
WHERE s.sale_date='2014-12-01' ORDER BY s.store_key, s.sale_date;

EnableForceOuter is on
Access Path:
+-SORT [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +--> JOIN HASH [Cost: 5K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Inner (BROADCAST)
| | Join Cond: (sales.product_key = products.product_key)
| | Materialize at Output: products.product_description
| | Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for products [Cost: 20, Rows: 60K (NO STATISTICS)] (PATH ID: 3)
| | | Projection: public.products_b0
| | | Force outer level: 8
| | | Materialize: products.product_key
| | | Execute on: All Nodes
| | | Runtime Filter: (SIP1(HashJoin): products.product_key)
| | +-- Inner -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 4)
| | | Projection: store.store_sales_b0
| | | Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | Filter: (sales.sale_date = '2014-12-01'::date)
| | | Execute on: All Nodes
```

Restrictions

Vertica ignores `force_outer` settings when it performs the following operations:

- Outer joins: Vertica generally respects OUTER JOIN clauses regardless of the force_outer settings of the joined tables.
- MERGE statement joins.
- Queries that include the SYNTACTIC_JOIN hint.
- Half-join queries such as SEMI JOIN.
- Joins to subqueries, where the subquery is always processed as having a force_outer setting of 5 regardless of the force_outer settings of the tables that are joined in the subquery. This setting determines a subquery's designation as inner or outer input relative to other join inputs. If two subqueries are joined, the optimizer determines which one is the inner input, and which one the outer.

Range Joins

Vertica provides performance optimizations for <, <=, >, >=, and BETWEEN predicates in join ON clauses. These optimizations are particularly useful when a column from one table is restricted to be in a range specified by two columns of another table.

Key Ranges

Multiple, consecutive key values can map to the same dimension values. Consider, for example, a table of IPv4 addresses and their owners. Because large subnets (ranges) of IP addresses could belong to the same owner, this dimension can be represented as:

```
=> CREATE TABLE ip_owners(  
    ip_start INTEGER,  
    ip_end INTEGER,  
    owner_id INTEGER);  
=> CREATE TABLE clicks(  
    ip_owners INTEGER,  
    dest_ip INTEGER);
```

A query that associates a click stream with its destination can use a join similar to the following, which takes advantage of the range optimization:

```
=> SELECT owner_id, COUNT(*) FROM clicks JOIN ip_owners  
    ON clicks.dest_ip BETWEEN ip_start AND ip_end  
    GROUP BY owner_id;
```

Requirements

Operators `<`, `<=`, `>`, `>=`, or `BETWEEN` must appear as top-level conjunctive predicates for range join optimization to be effective, as shown in the following examples:

`BETWEEN` as the only predicate:

```
=> SELECT COUNT(*) FROM fact JOIN dim
     ON fact.point BETWEEN dim.start AND dim.end;
```

Comparison operators as top-level predicates (within `AND`):

```
=> SELECT COUNT(*) FROM fact JOIN dim
     ON fact.point > dim.start AND fact.point < dim.end;
```

`BETWEEN` as a top-level predicate (within `AND`):

```
=> SELECT COUNT(*) FROM fact JOIN dim
     ON (fact.point BETWEEN dim.start AND dim.end) AND fact.c <> dim.c;
```

Query not optimized because `OR` is top-level predicate (disjunctive):

```
=> SELECT COUNT(*) FROM fact JOIN dim
     ON (fact.point BETWEEN dim.start AND dim.end) OR dim.end IS NULL;
```

Notes

- Expressions are optimized in range join queries in many cases.
- If range columns can have `NULL` values indicating that they are open-ended, it is possible to use range join optimizations by replacing nulls with very large or very small values:

```
=> SELECT COUNT(*) FROM fact JOIN dim
     ON fact.point BETWEEN NVL(dim.start, -1) AND NVL(dim.end, 100000000000);
```

- If there is more than one set of ranging predicates in the same `ON` clause, the order in which the predicates are specified might impact the effectiveness of the optimization:

```
=> SELECT COUNT(*) FROM fact JOIN dim
     ON fact.point1 BETWEEN dim.start1 AND dim.end1
     AND fact.point2 BETWEEN dim.start2 AND dim.end2;
```

The optimizer chooses the first range to optimize, so write your queries so that the range you most want optimized appears first in the statement.

- The use of the range join optimization is not directly affected by any characteristics of the physical schema; no schema tuning is required to benefit from the optimization.
- The range join optimization can be applied to joins without any other predicates, and to HASH or MERGE joins.
- To determine if an optimization is in use, search for RANGE in the EXPLAIN plan.

Query Optimization

When you submit a query to Vertica for processing, the Vertica query optimizer automatically chooses a set of operations to compute the requested result. These operations together are called a *query plan*. The choice of operations can significantly affect how many resources are needed to compute query results, and overall run-time performance. Optimal performance depends in great part on the projections that are available for a given query.

This section describes the different operations that the optimizer uses and how you can facilitate optimizer performance.

Note: Database response time depends on many factors. These include type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if response times are already short, or are not hardware bound.

Initial Process for Improving Query Performance

To optimize query performance, begin by performing the following tasks:

1. [Run Database Designer.](#)
2. [Check query events proactively.](#)
3. [Review the query plan.](#)

Run Database Designer

Database Designer creates a physical schema for your database that provides optimal query performance. The first time you run Database Designer, you should create a comprehensive design that includes relevant sample queries and data on which to base the design. If you develop performance issues later, consider loading additional queries that you run frequently and then rerunning Database Designer to create an incremental design.

When you run Database Designer, choose the option, Update Statistics. The Vertica query optimizer uses statistics about the data to create a query plan. Statistics help the optimizer determine:

- Multiple eligible projections to answer the query
- The best order in which to perform joins
- Data distribution algorithms, such as broadcast and resegmentation

If your statistics become out of date, run the Vertica function [ANALYZE_STATISTICS](#) function to update statistics for a schema, table, or columns. For more information, see [Collecting Database Statistics](#).

Check Query Events Proactively

The [QUERY_EVENTS](#) system table returns information on query planning, optimization, and execution events.

The `EVENT_TYPE` column provides various event types:

- Some event types are [informational](#).
- Others you should [review for possible corrective action](#).
- Several are [most important to address](#).

The following table lists informational event types.

Event Type	Description
SIP_FALLBACK	This optimization did not apply to this query type.
SMALL_MERGE_REPLACED	Vertica has chosen a more efficient way to access the data by replacing a merge.
STORAGE_CONTAINERS_ELIMINATED	Vertica has performed partition pruning for the purpose of optimization.
GROUPBY PUSHDOWN	(Message is Internal to Vertica.)
NO GROUPBY PUSHDOWN	(Message is Internal to Vertica.)
VALUE_TRUNCATED	A character value is too long.
GROUP_BY_PREPASS_FALLBACK	Vertica could not run an optimization. In-memory prepass is disabled. The projection may not be optimal.
MERGE_CONVERTED_TO_UNION	Vertica has converted a merge operator to a union operator due to the sort order of the multi-threaded storage access stream.
TRANSITIVE PREDICATE	<p>Vertica has optimized by adding predicates to joins where it makes logical sense to do so.</p> <p>For example, for the statement, <code>SELECT * FROM A, B WHERE A.a = B.a AND A.a = 1;</code> Vertica may add a predicate <code>B a = 1</code> as a filter for better storage access of table B.</p>
NODE PRUNING	Vertica has performed node pruning, which is similar to partition pruning, but at the

Event Type	Description
	node level.
SEQUENCE CACHE REFILLED	Vertica has refilled sequence cache.
OUTER OVERRIDE USED	For efficiency and optimization, Vertica has swapped the inner/outer tables in a join. Vertica used the smaller table as the inner table.
OUTER OVERRIDE NOT USED	Vertica found swapping inner/outer tables in a join unnecessary because the inner/outer tables were in good order. (For example, a smaller table was used in an inner join.)
EXTERNAL_PREDICATE_PUSHDOWN_NOT_SUPPORTED	Predicate pushdown for older Hive versions may not be supported. For more information, see Improving Query Performance for Data Stored in HDFS .
LibHDFS++ UNSUPPORTED OPERATION	Vertica accessed HDFS using the hdfs URL scheme, but the HDFS cluster uses an unsupported feature such as wire encryption or HTTPS_ONLY. Vertica fell back to WebHDFS.
LibHDFS++ FAILOVER RETRY	Vertica attempted to contact a Name Node on an HDFS cluster that uses High Availability Name Node and did not receive a response. Vertica retried with a different Name Node.

The following table lists event types that you should review for possible corrective action.

Event Type	Description	Action
GROUP_BY_SPILLED	This event type is typically related to a specific type of query, which you may need to adjust.	Identify the type of query and make adjustments accordingly. You may need to adjust resource pools, projections, or the amount of RAM available. Try

Event Type	Description	Action
		running the query on a cluster with no additional workload.
RESEGMENTED_MANY_ROWS	This event type is typically related to a specific type of query, which you may need to adjust.	Do projections need to be segmented in a different way to allow for join locality? Can you rewrite the query to filter out more rows at storage access time? (Typically, Vertica does so automatically through predicate pushdown.) Review your explain plan.
RLE_OVERRIDDEN	The average run counts are not large enough for Run Length Encoding (RLE). This event occurs with queries where the filtered results for certain columns do not work with RLE because cardinality is less than 10.	Review and rewrite your query, if necessary.
PREDICATE_OUTSIDE_HISTOGRAM	A predicate value you are trying to match does not exist in a set of possible values for a specific column. For example, you try to match a VARCHAR value WHERE mystring = "ABC<newline>". In this case, the newline character throws off the predicate matching optimizations.	Review and rewrite your query, if necessary.

The following table lists more critical event types, and recommends actions you can take to resolve issues.

Event Type	Description	Action
WOS_SPILL	WOS ran out of memory, and began spilling to ROS.	Try one or more of the following: <ul style="list-style-type: none"> • Adjust the load process to run bigger batches less frequently. • Give the WOS resource pool more memory. • Increase how often the Tuple Mover moveout and mergeout processes run for high ingest rates. (Adjustment of the moveout process helps avoid ROS issues.)
NO HISTOGRAM	Indicates a table does not have an updated column histogram.	Running the function ANALYZE_STATISTICS most often corrects this issue.
MEMORY LIMIT HIT	Indicates query complexity or, possibly, lack of available system memory.	Consider adjusting the <code>MAXMEMORYSIZE</code> and <code>PLANNEDCONCURRENCY</code> resource pools so that the optimizer has sufficient memory. On a heavily used system, this event may occur more frequently.
DELETE WITH NON OPTIMIZED PROJECTION	One or more projections do not have your delete filter column in their sort order, causing Vertica difficulty identifying rows to mark as deleted.	Add the delete filter column to the end of every projection sort order for your target delete table.

Event Type	Description	Action
JOIN_SPILLED	Vertica has spilled a join to disk. A join spill event slows down the subject query and all other queries as it consumes resources while using disk as virtual memory.	Try the following (in sequence): <ul style="list-style-type: none">• Review the explain plan. The query could be too ambitious, for example, cross joining two large tables.• Consider adding the query to a lower priority pool to reduce impact on other queries.• Create projections that allow for a merge join instead of a hash join.• Adjust the <code>PLANNEDCONCURRENCY</code> resource pool so that queries have more memory to execute.

Review the Query Plan

A *query plan* is a sequence of step-like paths that the Vertica query optimizer selects to access or alter information in your Vertica database. There are two ways to get information about the query plan:

- Run the [EXPLAIN](#) command. Each step (path) represents a single operation that the optimizer uses for its execution strategy.
- Query the [QUERY_PLAN_PROFILES](#) system table. This table provides detailed execution status for currently running queries. Output from the `QUERY_PLAN_PROFILES` table shows the real-time flow of data and the time and resources consumed for each path in each query plan.

Column Encoding

You can potentially make queries faster by changing column encoding. Encoding reduces the on-disk size of your data so the amount of I/O required for queries is reduced, resulting in faster execution times. Make sure all columns and projections included in the query use the correct data encoding. To do this, take the following steps:

1. Run Database Designer to create an incremental design. Database Designer implements the optimum encoding and projection design.
2. After creating the incremental design, update statistics using the [ANALYZE_STATISTICS](#) function.
3. Run [EXPLAIN](#) with one or more of the queries you submitted to the design to make sure it is using the new projections.

Alternatively, run [DESIGNER_DESIGN_PROJECTION_ENCODINGS](#) to re-evaluate the current encoding and update it if necessary.

Improving Column Compression

If you see slow performance or a large storage footprint with your [FLOAT](#) data, evaluate the data and your business needs to see if it can be contained in a [NUMERIC](#) column with a precision of 18 digits or less. Converting a [FLOAT](#) column to a [NUMERIC](#) column can improve data compression, reduce the on-disk size of your database, and improve performance of queries on that column.

When you define a [NUMERIC](#) data type, you specify the precision and the scale; [NUMERIC](#) data are exact representations of data. [FLOAT](#) data types represent variable precision and approximate values; they take up more space in the database.

Converting [FLOAT](#) columns to [NUMERIC](#) columns is most effective when:

- [NUMERIC](#) precision is 18 digits or less. Performance of [NUMERIC](#) data is fine-tuned for the common case of 18 digits of precision. Vertica recommends converting [FLOAT](#) columns to [NUMERIC](#) columns only if they require precision of 18 digits or less.
- [FLOAT](#) precision is bounded, and the values will all fall within a specified precision for a [NUMERIC](#) column. One example is monetary values like product prices or financial transaction amounts. For example, a column defined as [NUMERIC\(11,2\)](#) can accommodate

prices from 0 to a few million dollars and can store cents, and compresses more efficiently than a FLOAT column.

If you try to load a value into a NUMERIC column that exceeds the specified precision, Vertica returns an error and does not load the data. If you assign a value with more decimal digits than the specified scale, the value is rounded to match the specified scale and stored in that column.

See Also

[Numeric Data Types](#)

Using Run Length Encoding

When you run Database Designer, you can choose to optimize for loads, which minimizes database footprint. In this case, Database Designer applies encodings to columns to maximize query performance. [Encoding options](#) include run length encoding (RLE), which replaces sequences (runs) of identical values in a column with a set of pairs, where each pair represents the number of contiguous occurrences for a given value: (*occurrences*, *value*).

RLE is generally applicable to a column with low-cardinality, and where identical values are contiguous—typically, because table data is sorted on that column. For example, a customer profile table typically includes a gender column that contains values of F and M only. Sorting on gender ensures runs of F or M values that can be expressed as a set of two pairs: (*occurrences*, F) and (*occurrences*, M). So, given 8,147 occurrences of F and 7,956 occurrences of M, and a projection that is sorted primarily on gender, Vertica can apply RLE and store these values as a single set of two pairs: (8147, F) and (7956, M). Doing so reduces this projection's footprint and improves query performance.

Projections for Queries with Predicates

If your query contains one or more predicates, you can modify the projections to improve the query's performance, as described in the following two examples.

Queries That Use Date Ranges

This example shows how to encode data using RLE and change the projection sort order to improve the performance of a query that retrieves all data within a given date range.

Suppose you have a query that looks like this:

```
=> SELECT * FROM trades
    WHERE trade_date BETWEEN '2016-11-01' AND '2016-12-01';
```

To optimize this query, determine whether all of the projections can perform the SELECT operation in a timely manner. Run SELECT COUNT(*) statement for each projection, specifying the date range, and note the response time. For example:

```
=> SELECT COUNT(*) FROM [ projection_name ]
    WHERE trade_date BETWEEN '2016-11-01' AND '2016-12-01';
```

If one or more of the queries is slow, check the uniqueness of the trade_date column and determine if it needs to be in the projection's ORDER BY clause and/or can be encoded using RLE. RLE replaces sequences of the same data values within a column by a pair that represents the value and a count. For best results, order the columns in the projection from lowest cardinality to highest cardinality, and use RLE to encode the data in low-cardinality columns.

Note: For an example of using sorting and RLE, see [Combine RLE and Sort Order](#).

If the number of unique columns is unsorted, or if the average number of repeated rows is less than 10, trade_date is too close to being unique and cannot be encoded using RLE. In this case, add a new column to minimize the search scope.

The following example adds a new column trade_year:

1. Determine if the new column trade_year returns a manageable result set. The following query returns the data grouped by trade_year:

```
=> SELECT DATE_TRUNC('trade_year', trade_date), COUNT(*)
    FROM trades
    GROUP BY DATE_TRUNC('trade_year', trade_date);
```

2. Assuming that trade_year = 2007 is near 8k, add a column for trade_year to the trades table. The SELECT statement then becomes:

```
=> SELECT * FROM trades
    WHERE trade_year = 2007
```

```
AND trade_date BETWEEN '2016-11-01' AND '2016-12-01';
```

As a result, you have a projection that is sorted on `trade_year`, which can be encoded using RLE.

Queries for Tables with a High-Cardinality Primary Key

This example demonstrates how you can modify the projection to improve the performance of queries that select data from a table with a high-cardinality primary key.

Suppose you have the following query:

```
=> SELECT FROM [table]
WHERE pk IN (12345, 12346, 12347,...);
```

Because the primary key is a high-cardinality column, Vertica has to search a large amount of data.

To optimize the schema for this query, create a new column named `buckets` and assign it the value of the primary key divided by 10000. In this example, `buckets=(int) pk/10000`. Use the `buckets` column to limit the search scope as follows:

```
=> SELECT FROM [table]
WHERE buckets IN (1,...)
AND pk IN (12345, 12346, 12347,...);
```

Creating a lower cardinality column and adding it to the query limits the search scope and improves the query performance. In addition, if you create a projection where `buckets` is first in the sort order, the query may run even faster.

GROUP BY Queries

The following sections include several examples that show how you can design your projections to optimize the performance of your GROUP BY queries.

GROUP BY Implementation Options

Vertica implements a query GROUP BY clause with one of these algorithms: GROUPBY PIPELINED or GROUPBY HASH. Both algorithms return the same results. Performance of both is generally similar for queries that return a small number of distinct groups—typically a thousand per node .

You can use [EXPLAIN](#) to determine which algorithm the query optimizer chooses for a given query. The following conditions generally determine which algorithm is chosen:

- GROUPBY PIPELINED requires all GROUP BY data to be specified in the projection's ORDER BY clause. For details, see [GROUPBY PIPELINED Requirements](#) below.

Because GROUPBY PIPELINED only needs to retain in memory the current group data, this algorithm generally requires less memory and executes faster than GROUPBY HASH. Performance improvements are especially notable for queries that aggregate large numbers of distinct groups.

- GROUPBY HASH is used for any query that does not comply with GROUP BY PIPELINED sort order requirements. In this case, Vertica must build a hash table on GROUP BY columns before it can start grouping the data.

GROUPBY PIPELINED Requirements

You can enable use of the GROUP BY PIPELINED algorithm by ensuring that the query and one of its projections comply with GROUP BY PIPELINED requirements. The following conditions apply to GROUPBY PIPELINED. If none of them is true for the query, then Vertica uses GROUPBY HASH.

All examples that follow assume this schema:

```
CREATE TABLE sortopt (  
  a INT NOT NULL,  
  b INT NOT NULL,
```

```

        c INT,
        d INT
    );
CREATE PROJECTION sortopt_p (
    a_proj,
    b_proj,
    c_proj,
    d_proj )
AS SELECT * FROM sortopt
ORDER BY a,b,c
UNSEGMENTED ALL NODES;
INSERT INTO sortopt VALUES(5,2,13,84);
INSERT INTO sortopt VALUES(14,22,8,115);
INSERT INTO sortopt VALUES(79,9,401,33);

```

Condition 1

All GROUP BY columns are also included in the projection ORDER BY clause.

For example:

GROUP BY columns	GROUPBY algorithm	Reason chosen
a a, b b, a a, b, c c, a, b	GROUPBY PIPELINED	Columns a, b, and c are included in the projection sort columns.
a, b, c, d	GROUPBY HASH	Column d is not part of the projection sort columns.

Condition 2

If the query's GROUP BY clause has fewer columns than the projection's ORDER BY clause, the GROUP BY columns must:

- Be a subset of ORDER BY columns that are contiguous.
- Include the first ORDER BY column.

For example:

GROUP BY columns	GROUPBY algorithm	Reason chosen
a a, b b, a	GROUPBY PIPELINED	All GROUP BY columns are a subset of contiguous columns in the projection's ORDER BY clause {a, b, c}, and include column a.

GROUP BY columns	GROUPBY algorithm	Reason chosen
a, c	GROUPBY HASH	GROUP BY columns {a, c} are not contiguous in the projection ORDER BY clause {a, b, c}.
b, c	GROUPBY HASH	GROUP BY columns {b, c} do not include the projection's first ORDER BY column a.

Condition 3

If a query's GROUP BY columns do not appear first in the projection's ORDER BY clause, then any early-appearing projection sort columns that are missing in the query's GROUP BY clause must be present as single-column constant equality predicates in the query's WHERE clause.

For example:

Query	GROUPBY algorithm	Reason chosen
SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY b	GROUPBY PIPELINED	All columns preceding b in the projection sort order appear as constant equality predicates.
SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY a, b	GROUPBY PIPELINED	Grouping column a is redundant but has no effect on algorithm selection.
SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY b, c	GROUPBY PIPELINED	All columns preceding b and c in the projection sort order appear as constant equality predicates.
SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY c, b	GROUPBY PIPELINED	All columns preceding b and c in the projection sort order appear as constant equality predicates.
SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY c	GROUPBY HASH	All columns preceding c in the projection sort order do not appear as constant equality predicates.

Controlling GROUPBY Algorithm Choice

It is generally best to allow Vertica to determine which GROUP BY algorithm is best suited for a given query. Occasionally, you might want to use one algorithm over another. In such cases,

you can qualify the GROUP BY clause with a [GBYTYPE](#) hint:

```
GROUP BY /*+ GBYTYPE( HASH | PIPE ) */
```

For example, given the following query, the query optimizer uses the GROUPBY PIPELINED algorithm:

```
=> EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY a,b;
-----
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY a,b;

Access Path:
+-GROUPBY PIPELINED [Cost: 11, Rows: 3 (NO STATISTICS)] (PATH ID: 1)
| Aggregates: sum(sortopt.a)
| Group By: sortopt.a, sortopt.b

...
```

You can use the GBYTYPE hint to force the query optimizer to use the GROUPBY HASH algorithm instead:

```
=> EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY /*+GBYTYPE(HASH) */ a,b;
-----
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY /*+GBYTYPE(HASH) */ a,b;

Access Path:
+-GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 11, Rows: 3 (NO STATISTICS)] (PATH ID: 1)
| Aggregates: sum(sortopt.a)
| Group By: sortopt.a, sortopt.b

...
```

The GBYTYPE hint can specify a PIPE (GROUPBY PIPELINED algorithm) argument only if the query and one of its projections comply with GROUP BY PIPELINED [requirements](#). Otherwise, Vertica issues a warning and uses GROUPBY HASH.

For example, the following query cannot use the GROUPBY PIPELINED algorithm, as the GROUP BY columns {b, c} do not include the projection's first ORDER BY column a:

```
=> SELECT SUM(a) FROM sortopt GROUP BY /*+GBYTYPE(PIPE) */ b,c;
WARNING 7765: Cannot apply Group By Pipe algorithm. Proceeding with Group By Hash and hint will be ignored
SUM
-----
 79
 14
  5
(3 rows)
```

Avoiding Resegmentation During GROUP BY Optimization with Projection Design

To compute the correct result of a query that contains a GROUP BY clause, Vertica must ensure that all rows with the same value in the GROUP BY expressions end up at the same node for final computation. If the projection design already guarantees the data is segmented by the GROUP BY columns, no resegmentation is required at run time.

To avoid resegmentation, the GROUP BY clause must contain all the segmentation columns of the projection, but it can also contain other columns.

When your query includes a GROUP BY clause and joins, if the join depends on the results of the GROUP BY, as in the following example, Vertica performs the GROUP BY first:

```
=> EXPLAIN SELECT * FROM (SELECT b from foo GROUP BY b) AS F, foo WHERE foo.a = F.b;

Access Path:
+-JOIN MERGEJOIN(inputs presorted) [Cost: 649, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
|   Join Cond: (foo.a = F.b)
|   Materialize at Output: foo.b
|   Execute on: All Nodes
|   +- Outer -> STORAGE ACCESS for foo [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
|   |   Projection: public.foo_super
|   |   Materialize: foo.a
|   |   Execute on: All Nodes
|   |   Runtime Filter: (SIP1(MergeJoin): foo.a)
|   +- Inner -> SELECT [Cost: 245, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
|   |   Execute on: All Nodes
|   |   +---> GROUPBY HASH (SORT OUTPUT) (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 245,
Rows: 10K (NO STATISTICS)] (PATH ID:
4)
|   |   |   Group By: foo.b
|   |   |   Execute on: All Nodes
|   |   +---> STORAGE ACCESS for foo [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 5)
|   |   |   Projection: public.foo_super
|   |   |   Materialize: foo.b
|   |   |   Execute on: All Nodes
```

If the result of the join operation is the input to the GROUP BY clause, Vertica performs the join first, as in the following example. The segmentation of those intermediate results may not be consistent with the GROUP BY clause in your query, resulted in resegmentation at run time.

```
=> EXPLAIN SELECT * FROM foo AS F, foo WHERE foo.a = F.b GROUP BY 1,2,3,4;

Access Path:
+-GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 869, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
|   Group By: F.a, F.b, foo.a, foo.b
|   Execute on: All Nodes
|   +---> JOIN HASH [Cost: 853, Rows: 10K (NO STATISTICS)] (PATH ID: 2) Outer (RESEGMENT)(LOCAL ROUND
ROBIN)
|   |   Join Cond: (foo.a = F.b)
```

```

| |      Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for F [Cost: 403, Rows: 10K (NO STATISTICS)] (PUSHED GROUPING) (PATH
ID: 3)
| | |      Projection: public.foo_super
| | |      Materialize: F.a, F.b
| | |      Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for foo [Cost: 403, Rows: 10K (NO STATISTICS)] (PATH ID: 4)
| | |      Projection: public.foo_super
| | |      Materialize: foo.a, foo.b
| | |      Execute on: All Nodes

```

If your query does not include joins, the GROUP BY clauses are processed using the existing database projections.

Examples

Assume the following projection:

```
CREATE PROJECTION ... SEGMENTED BY HASH(a,b) ALL NODES
```

The following table explains whether or not resegmentation occurs at run time and why.

GROUP BY a	Requires resegmentation at run time. The query does not contain all the projection segmentation columns.
GROUP BY a, b	Does not require resegmentation at run time. The GROUP BY clause contains all the projection segmentation columns.
GROUP BY a, b, c	Does not require resegmentation at run time. The GROUP BY clause contains all the projection segmentation columns.
GROUP BY a+1, b	Requires resegmentation at run time because of the expression on column a.

To determine if resegmentation will occur during your GROUP BY query, look at the [EXPLAIN](#)-generated query plan.

For example, the following plan uses GROUPBY PIPELINED sort optimization and requires resegmentation to perform the GROUP BY calculation:

```
+--GROUPBY PIPELINED (RESEGMENT GROUPS) [Cost: 194, Rows: 10K (NO
STATISTICS)] (PATH ID: 1)
```

The following plan uses GROUPBY PIPELINED sort optimization, but does not require resegmentation:

```
+--GROUPBY PIPELINED [Cost: 459, Rows: 10K (NO STATISTICS)] (PATH ID:
1)
```

DISTINCT in a SELECT Query List

This section describes how to optimize queries that have the `DISTINCT` keyword in their `SELECT` list. The techniques for optimizing `DISTINCT` queries are similar to the techniques for optimizing `GROUP BY` queries because when processing queries that use `DISTINCT`, the Vertica optimizer rewrites the query as a `GROUP BY` query.

The following sections below this page describe specific situations:

- [Query Has No Aggregates in SELECT List](#)
- [COUNT \(DISTINCT\) and Other DISTINCT Aggregates](#)
- [Approximate Count Distinct Functions](#)
- [Single DISTINCT Aggregates](#)
- [Multiple DISTINCT Aggregates](#)

Examples in these sections use the following table:

```
=> CREATE TABLE table1 (  
    a INT,  
    b INT,  
    c INT  
);
```

Query Has No Aggregates in SELECT List

If your query has no aggregates in the `SELECT` list, internally, Vertica treats the query as if it uses `GROUP BY` instead.

For example, you can rewrite the following query:

```
SELECT DISTINCT a, b, c FROM table1;
```

as:

```
SELECT a, b, c FROM table1 GROUP BY a, b, c;
```

For fastest execution, apply the optimization techniques for `GROUP BY` queries described in [GROUP BY Queries](#).

COUNT (DISTINCT) and Other DISTINCT Aggregates

Computing a DISTINCT aggregate generally requires more work than other aggregates. Also, a query that uses a single DISTINCT aggregate consumes fewer resources than a query with multiple DISTINCT aggregates.

Tip: Vertica executes queries with multiple distinct aggregates more efficiently when all distinct aggregate columns have a similar number of distinct values.

Approximate Count Distinct Functions

Vertica provides the COUNT(DISTINCT) function to compute the exact number of distinct values in a data set. If projections are available that allow COUNT(DISTINCT) to execute using the GROUPBY PIPELINED algorithm, COUNT(DISTINCT) performs well. In some situations, however, using APPROXIMATE_COUNT_DISTINCT performs better than COUNT(DISTINCT).

A [COUNT \[Aggregate\]](#) operation performs well when:

- One of the sorted projections delivers an order that enables sorted aggregation to be performed.
- The number of distinct values is fairly small.
- Hashed aggregation is required to execute the query.

When an approximate value will suffice or the values need to be rolled up, consider using the APPROXIMATE_COUNT_DISTINCT* functions.

Note: The APPROXIMATE_COUNT_DISTINCT* functions cannot appear in the same query block as DISTINCT aggregates.

Use Cases

Use [APPROXIMATE_COUNT_DISTINCT](#) as a direct replacement for COUNT (DISTINCT) when:

- You have a large data set and you do not require an exact count of distinct values.
- The performance of COUNT(DISTINCT) on a given data set is insufficient.

- You calculate several distinct counts in the same query.
- The plan for COUNT(DISTINCT) uses hashed aggregation.

Most of the time, APPROXIMATE_COUNT_DISTINCT executes faster than a comparable COUNT (DISTINCT) operation when hashed.

The expected value that APPROXIMATE_COUNT_DISTINCT returns is equal to COUNT (DISTINCT), with an error that is lognormally distributed with standard deviation s . You can control the standard deviation directly by setting the *error_tolerance*.

Use [APPROXIMATE_COUNT_DISTINCT_SYNOPSIS](#) and [APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS](#) together when:

- You have a large data set and you don't require an exact count of distinct values.
- The performance of COUNT(DISTINCT) on a given data set is insufficient.

and

- You want to pre-compute the distinct counts and later combine them in different ways.

Pass APPROXIMATE_COUNT_DISTINCT_SYNOPSIS the data set and a normally distributed confidence interval. The function returns a subset of the data, called a *synopsis*.

Pass the synopsis to the APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS function, which then performs an approximate count distinct on the synopsis.

Single DISTINCT Aggregates

Vertica computes a DISTINCT aggregate by first removing all duplicate values of the aggregate's argument to find the distinct values. Then it computes the aggregate.

For example, you can rewrite the following query:

```
SELECT a, b, COUNT(DISTINCT c) AS dcnt FROM table1 GROUP BY a, b;
```

as:

```
SELECT a, b, COUNT(dcnt) FROM  
(SELECT a, b, c AS dcnt FROM table1 GROUP BY a, b, c)  
GROUP BY a, b;
```

For fastest execution, apply the optimization techniques for GROUP BY queries.

Multiple DISTINCT Aggregates

If your query has multiple `DISTINCT` aggregates, there is no straightforward SQL rewrite that can compute them. The following query cannot easily be rewritten for improved performance:

```
SELECT a, COUNT(DISTINCT b), COUNT(DISTINCT c) AS dcnt FROM table1 GROUP BY a;
```

For a query with multiple `DISTINCT` aggregates, there is no projection design that can avoid using `GROUPBY HASH` and resegmenting the data. To improve performance of this query, make sure that it has large amounts of memory available. For more information about memory allocation for queries, see [Resource Manager](#).

JOIN Queries

In general, you can optimize execution of queries that join multiple tables in several ways:

- [Create projections for the joined tables that are sorted on join predicate columns.](#) This facilitates use of the merge join algorithm, which generally joins tables more efficiently than the merge join algorithm.
- [Create projections that are identically segmented on the join keys.](#)

Other Best Practices

Vertica also executes joins more efficiently if the following conditions are true:

- Query construction enables the query optimizer to create a plan where the larger table is defined as the outer input.
- The columns on each side of the equality predicate are from the same table. For example in the following query, the left and right sides of the equality predicate include only columns from tables T and X, respectively:

```
=> SELECT * FROM T JOIN X ON T.a + T.b = X.x1 - X.x2;
```

Conversely, the following query incurs more work to process, because the right side of the predicate includes columns from both tables T and X:

```
=> SELECT * FROM T JOIN X WHERE T.a = X.x1 + T.b
```

Hash Joins Versus Merge Joins

The Vertica optimizer implements a join with one of the following algorithms:

- **Merge join** is used when projections of the joined tables are sorted on the join columns. Merge joins are faster and uses less memory than hash joins.
- **Hash join** is used when projections of the joined tables are not already sorted on the join columns. In this case, the optimizer builds an in-memory hash table on the inner table's join column. The optimizer then scans the outer table for matches to the hash table, and joins

data from the two tables accordingly. The cost of performing a hash join is low if the entire hash table can fit in memory. Cost rises significantly if the hash table must be written to disk.

The optimizer automatically chooses the most appropriate algorithm to execute a query, given the projections that are available.

Facilitating Merge Joins

To facilitate a merge join, create projections for the joined tables that are sorted on the join predicate columns. The join predicate columns should be the first columns in the `ORDER BY` clause.

For example, tables `first` and `second` are defined as follows, with projections `first_p1` and `second_p1`, respectively. The projections are sorted on `data_first` and `data_second`:

```
CREATE TABLE first ( id INT, data_first INT );
CREATE PROJECTION first_p1 AS SELECT * FROM first ORDER BY data_first;

CREATE TABLE second ( id INT, data_second INT );
CREATE PROJECTION second_p1 AS SELECT * FROM first ORDER BY data_second;
```

When you join these tables on unsorted columns `first.id` and `second.id`, Vertica uses the hash join algorithm:

```
EXPLAIN SELECT first.data_first, second.data_second FROM first JOIN second ON first.id = second.id;

Access Path:
+-JOIN HASH [Cost: 752, Rows: 300K] (PATH ID: 1) Inner (BROADCAST)
```

You can facilitate execution of this query with the merge join algorithm by creating projections `first_p2` and `second_p2`, which are sorted on join columns `first_p2.id` and `second_p2.id`, respectively:

```
CREATE PROJECTION first_p2 AS SELECT id, data_first FROM first ORDER BY id SEGMENTED BY hash(id,
data_first) ALL NODES;
CREATE PROJECTION second_p2 AS SELECT id, data_second FROM second ORDER BY id SEGMENTED BY hash(id,
data_second) ALL NODES;
```

If the query joins significant amounts of data, the query optimizer uses the merge algorithm:

```
EXPLAIN SELECT first.data_first, second.data_second FROM first JOIN second ON first.id = second.id;

Access Path:
+-JOIN MERGEJOIN(inputs presorted) [Cost: 731, Rows: 300K] (PATH ID: 1) Inner (BROADCAST)
```

You can also facilitate a merge join by using subqueries to pre-sort the join predicate columns. For example:

```
SELECT first.id, first.data_first, second.data_second FROM  
  (SELECT * FROM first ORDER BY id ) first JOIN (SELECT * FROM second ORDER BY id) second ON first.id  
  = second.id;
```

Identical Segmentation

To improve query performance when you join multiple tables, create projections that are identically segmented on the join keys. Identically-segmented projections allow the joins to occur locally on each node, thereby helping to reduce data movement across the network during query processing.

To determine if projections are identically-segmented on the query join keys, create a query plan with EXPLAIN. If the query plan contains RESEGMENT or BROADCAST, the projections are not identically segmented.

The Vertica optimizer chooses a projection to supply rows for each table in a query. If the projections to be joined are segmented, the optimizer evaluates their segmentation against the query join expressions. It thereby determines whether the rows are placed on each node so it can join them without fetching data from another node.

Join Conditions for Identically Segmented Projections

A projection *p* is segmented on join columns if all column references in *p*'s segmentation expression are a subset of the columns in the join expression.

The following conditions must be true for two segmented projections *p1* of table *t1* and *p2* of table *t2* to participate in a join of *t1* to *t2*:

- The join condition must have the following form:

```
t1.j1 = t2.j1 AND t1.j2 = t2.j2 AND ... t1.jN = t2.jN
```

- The join columns must share the same base data type. For example:
 - If *t1.j1* is an INTEGER, *t2.j1* can be an INTEGER but it cannot be a FLOAT.
 - If *t1.j1* is a CHAR(10), *t2.j1* can be any CHAR or VARCHAR (for example, CHAR(10), VARCHAR(10), VARCHAR(20)), but *t2.j1* cannot be an INTEGER.
- If *p1* is segmented by an expression on columns {*t1.s1*, *t1.s2*, ... *t1.sN*}, each segmentation column *t1.sX* must be in the join column set {*t1.jX*}.

- If p2 is segmented by an expression on columns {t2.s1, t2.s2, ... t2.sN}, each segmentation column t2.sX must be in the join column set {t2.jX}.
- The segmentation expressions of p1 and p2 must be structurally equivalent. For example:
 - If p1 is SEGMENTED BY hash(t1.x) and p2 is SEGMENTED BY hash(t2.x), p1 and p2 are identically segmented.
 - If p1 is SEGMENTED BY hash(t1.x) and p2 is SEGMENTED BY hash(t2.x + 1), p1 and p2 are not identically segmented.
- p1 and p2 must have the same segment count.
- The assignment of segments to nodes must match. For example, if p1 and p2 use an OFFSET clause, their offsets must match.
- If Vertica finds projections for t1 and t2 that are not identically segmented, the data is redistributed across the network during query run time, as necessary.

Tip: If you create custom designs, try to use segmented projections for joins whenever possible.

Examples

The following statements create two tables and specify to create identical segments:

```
=> CREATE TABLE t1 (id INT, x1 INT, y1 INT) SEGMENTED BY HASH(id, x1) ALL NODES;  
=> CREATE TABLE t2 (id INT, x1 INT, y1 INT) SEGMENTED BY HASH(id, x1) ALL NODES;
```

Given this design, the join conditions in the following queries can leverage identical segmentation:

```
=> SELECT * FROM t1 JOIN t2 ON t1.id = t2.id;  
=> SELECT * FROM t1 JOIN t2 ON t1.id = t2.id AND t1.x1 = t2.x1;
```

Conversely, the join conditions in the following queries require resegmentation:

```
=> SELECT * FROM t1 JOIN t2 ON t1.x1 = t2.x1;  
=> SELECT * FROM t1 JOIN t2 ON t1.id = t2.x1;
```

See Also

- [Comparing Partitioning and Segmentation](#)
- [CREATE PROJECTION](#)

ORDER BY Queries

You can improve the performance of queries that contain only `ORDER BY` clauses if the columns in a projection's `ORDER BY` clause are the same as the columns in the query.

If you define the projection sort order in the `CREATE PROJECTION` statement, the Vertica query optimizer does not have to sort projection data before performing certain `ORDER BY` queries.

The following table, `sortopt`, contains the columns `a`, `b`, `c`, and `d`. Projection `sortopt_p` specifies to order on columns `a`, `b`, and `c`.

```
CREATE TABLE sortopt (  
  a INT NOT NULL,  
  b INT NOT NULL,  
  c INT,  
  d INT  
);  
CREATE PROJECTION sortopt_p (  
  a_proj,  
  b_proj,  
  c_proj,  
  d_proj )  
AS SELECT * FROM sortopt  
ORDER BY a,b,c  
UNSEGMENTED ALL NODES;  
INSERT INTO sortopt VALUES(5,2,13,84);  
INSERT INTO sortopt VALUES(14,22,8,115);  
INSERT INTO sortopt VALUES(79,9,401,33);
```

Based on this sort order, if a `SELECT * FROM sortopt` query contains one of the following `ORDER BY` clauses, the query does not have to resort the projection:

- `ORDER BY a`
- `ORDER BY a, b`
- `ORDER BY a, b, c`

For example, Vertica does not have to resort the projection in the following query because the sort order includes columns specified in the `CREATE PROJECTION .ORDER BY a, b, c` clause, which mirrors the query's `ORDER BY a, b, c` clause:

```
=> SELECT * FROM sortopt ORDER BY a, b, c;  
 a | b | c | d  
----+-----+-----+-----  
 5 | 2 | 13 | 84  
14 | 22 | 8 | 115
```

```
79 | 9 | 401 | 33  
(3 rows)
```

If you include column `d` in the query, Vertica must re-sort the projection data because column `d` was not defined in the `CREATE PROJECTION . .ORDER BY` clause. Therefore, the `ORDER BY d` query won't benefit from any sort optimization.

You cannot specify an `ASC` or `DESC` clause in the `CREATE PROJECTION` statement's `ORDER BY` clause. Vertica always uses an ascending sort order in physical storage, so if your query specifies descending order for any of its columns, the query still causes Vertica to re-sort the projection data. For example, the following query requires Vertica to sort the results:

```
=> SELECT * FROM sortopt ORDER BY a DESC, b, c;  
a | b | c | d  
-----  
79 | 9 | 401 | 33  
14 | 22 | 8 | 115  
5 | 2 | 13 | 84  
(3 rows)
```

See Also

[CREATE PROJECTION](#)

Analytic Functions

The following sections describe how to optimize SQL-99 analytic functions that Vertica supports.

Empty OVER Clauses

The `OVER()` clause does not require a windowing clause. If your query uses an analytic function like `SUM(x)` and you specify an empty `OVER()` clause, the analytic function is used as a reporting function, where the entire input is treated as a single partition; the aggregate returns the same aggregated value for each row of the result set. The query executes on a single node, potentially resulting in poor performance.

If you add a `PARTITION BY` clause to the `OVER()` clause, the query executes on multiple nodes, improving its performance.

NULL Sort Order

By default, projection column values are stored in ascending order, but placement of NULL values depends on a column's data type.

NULL Placement Differences With ORDER BY Clauses

The analytic `OVER(window-order-clause)` and the SQL `ORDER BY` clause have slightly different semantics:

OVER(ORDER BY ...)

The analytic window order clause uses the `ASC` or `DESC` sort order to determine `NULLS FIRST` or `NULLS LAST` placement for analytic function results. NULL values are placed as follows:

- `ASC, NULLS LAST` — NULL values appear at the end of the sorted result.
- `DESC, NULLS FIRST` — NULL values appear at the beginning of the sorted result.

(SQL) ORDER BY

The SQL and VERTICA ORDER BY clauses produce different results. The SQL ORDER BY clause specifies only ascending or descending sort order. The VERTICA ORDER BY clause determines NULL placement based on the column data type:

- NUMERIC, INTEGER, DATE, TIME, TIMESTAMP, and INTERVAL columns: NULLS FIRST (NULL values appear at the beginning of a sorted projection.)
- FLOAT, STRING, and BOOLEAN columns: NULLS LAST (NULL values appear at the end of a sorted projection.)

NULL Sort Options

If you do not care about NULL placement in queries that involve analytic computations, or if you know that columns do not contain any NULL values, specify NULLS AUTO—irrespective of data type. Vertica chooses the placement that gives the fastest performance, as in the following query. Otherwise, specify NULLS FIRST or NULLS LAST.

```
=> SELECT x, RANK() OVER (ORDER BY x NULLS AUTO) FROM t;
```

You can carefully formulate queries so Vertica can avoid sorting the data and increase query performance, as illustrated by the following example. Vertica sorts inputs from table `t` on column `x`, as specified in the `OVER(ORDER BY)` clause, and then evaluates `RANK()`:

```
=> CREATE TABLE t (  
  x FLOAT,  
  y FLOAT );  
=> CREATE PROJECTION t_p (x, y) AS SELECT * FROM t  
  ORDER BY x, y UNSEGMENTED ALL NODES;  
=> SELECT x, RANK() OVER (ORDER BY x) FROM t;
```

In the preceding `SELECT` statement, Vertica eliminates the `ORDER BY` clause and executes the query quickly because column `x` is a `FLOAT` data type. As a result, the projection sort order matches the analytic default ordering (`ASC + NULLS LAST`). Vertica can also avoid having to sort the data when the underlying projection is already sorted.

However, if column `x` is an `INTEGER` data type, Vertica must sort the data because the projection sort order for `INTEGER` data types (`ASC + NULLS FIRST`) does not match the default analytic ordering (`ASC + NULLS LAST`). To help Vertica eliminate the sort, specify the placement of `NULLS` to match the default ordering:

```
=> SELECT x, RANK() OVER (ORDER BY x NULLS FIRST) FROM t;
```

If column `x` is a `STRING`, the following query eliminates the sort:

```
=> SELECT x, RANK() OVER (ORDER BY x NULLS LAST) FROM t;
```

If you omit `NULLS LAST` in the preceding query, VERTICA eliminates the sort because `ASC + NULLS LAST` is the default sort specification for both the analytic `ORDER BY` clause and for string-related columns in Vertica.

See Also

- [Runtime Sorting of NULL Values in Analytic Functions](#)
- [SQL Analytics](#)

Runtime Sorting of NULL Values in Analytic Functions

By carefully writing queries or creating your design (or both), you can help the Vertica query optimizer skip sorting all columns in a table when performing an analytic function, which can improve query performance.

To minimize Vertica's need to sort projections during query execution, redefine the `employee` table and specify that `NULL` values are not allowed in the sort fields:

```
=> DROP TABLE employee CASCADE;
=> CREATE TABLE employee
  (empno INT,
   deptno INT NOT NULL,
   sal INT NOT NULL);
CREATE TABLE
=> CREATE PROJECTION employee_p AS
  SELECT * FROM employee
  ORDER BY deptno, sal;
CREATE PROJECTION
=> INSERT INTO employee VALUES(101,10,50000);
=> INSERT INTO employee VALUES(103,10,43000);
=> INSERT INTO employee VALUES(104,10,45000);
=> INSERT INTO employee VALUES(105,20,97000);
=> INSERT INTO employee VALUES(108,20,33000);
=> INSERT INTO employee VALUES(109,20,51000);
=> COMMIT;
COMMIT
```

```
=> SELECT * FROM employee;
 empno | deptno | sal
-----+-----+-----
   101 |      10 | 50000
   103 |      10 | 43000
   104 |      10 | 45000
   105 |      20 | 97000
   108 |      20 | 33000
   109 |      20 | 51000
(6 rows)
=> SELECT deptno, sal, empno, RANK() OVER
```

```
(PARTITION BY deptno ORDER BY sal)
FROM employee;
deptno | sal | empno | ?column?
-----+-----+-----+-----
 10 | 43000 | 103 | 1
 10 | 45000 | 104 | 2
 10 | 50000 | 101 | 3
 20 | 33000 | 108 | 1
 20 | 51000 | 109 | 2
 20 | 97000 | 105 | 3
(6 rows)
```

Tip: If you do not care about NULL placement in queries that involve analytic computations, or if you know that columns contain no NULL values, specify `NULLS AUTO` in your queries. Vertica attempts to choose the placement that gives the fastest performance. Otherwise, specify `NULLS FIRST` or `NULLS LAST`.

LIMIT Queries with ROW_NUMBER Predicates

Queries that use the [LIMIT clause](#) with `ORDER BY` or analytic function `ROW_NUMBER()` return a specific subset of rows in the query result. Vertica processes these queries efficiently using *Top-K Optimization*, which is a database query ranking process. Top-K optimization avoids sorting (and potentially writing to disk) an entire data set to find a small number of rows. This can significantly improve query performance.

For example, in the following query, Vertica extracts only the three smallest rows from column `x`:

```
=> SELECT * FROM t1 ORDER BY x LIMIT 3;
```

If table `t1` contains millions of rows, it is time consuming to sort all the `x` values. Instead, Vertica keeps track of the smallest three values in `x`.

Note: If you omit the `ORDER BY` clause, when using the `LIMIT` clause, the results can be nondeterministic.

Sort operations that precede a [SQL analytics](#) computation benefit from Top-K optimization if the query contains an `OVER(ORDER BY)` clause and a predicate on the `ROW_NUMBER` function, as in the following example:

```
=> SELECT x FROM
  (SELECT *, ROW_NUMBER() OVER (ORDER BY x) AS row
   FROM t1) t2 WHERE row <= 3;
```

The preceding query has the same behavior as the following query, which uses a `LIMIT` clause:

```
=> SELECT ROW_NUMBER() OVER (ORDER BY x) AS RANK FROM t1 LIMIT 3;
```

You can use `ROW_NUMBER()` with the analytic [window partition clause](#), something you cannot do if you use `LIMIT`:

```
=> SELECT x, y FROM  
      (SELECT *, ROW_NUMBER() OVER (PARTITION BY x ORDER BY y)  
       AS row FROM t1) t2 WHERE row <= 3;
```

Note: When the `OVER()` clause includes the *window-partition-clause*, Top-K optimization occurs only when the analytic sort matches the input's sort, for example, if the projection is sorted on columns `x` and `y` in table `t1`.

If you still want to improve the performance of your query, consider using the optimization techniques described in [ORDER BY Queries](#).

INSERT-SELECT Operations

There are several ways to optimize an INSERT-SELECT query that has the following format:

```
INSERT /*+direct*/ INTO destination SELECT * FROM source;
```

Matching Sort Orders

When performing INSERT-SELECT operations, to avoid the sort phase of the INSERT, make sure that the sort order for the SELECT query matches the projection sort order of the target table.

For example, on a single-node database:

```
=> CREATE TABLE source (col1 INT, col2 INT, col3 INT);
=> CREATE PROJECTION source_p (col1, col2, col3)
  AS SELECT col1, col2, col3 FROM source
  ORDER BY col1, col2, col3
  SEGMENTED BY HASH(col3)
  ALL NODES;
=> CREATE TABLE destination (col1 INT, col2 INT, col3 INT);
=> CREATE PROJECTION destination_p (col1, col2, col3)
  AS SELECT col1, col2, col3 FROM destination
  ORDER BY col1, col2, col3
  SEGMENTED BY HASH(col3)
  ALL NODES;
```

The following INSERT does not require a sort because the query result has the column order of the projection:

```
=> INSERT /*+direct*/ INTO destination SELECT * FROM source;
```

The following INSERT requires a sort because the order of the columns in the SELECT statement does not match the projection order:

```
=> INSERT /*+direct*/ INTO destination SELECT col1, col3, col2 FROM source;
```

The following INSERT does not require a sort. The order of the columns doesn't match, but the explicit ORDER BY causes the output to be sorted by c1, c3, c2 in Vertica:

```
=> INSERT /*+direct*/ INTO destination SELECT col1, col3, col2 FROM source
  GROUP BY col1, col3, col2
  ORDER BY col1, col2, col3 ;
```

Identical Segmentation

When performing an INSERT-SELECT operation from a segmented source table to a segmented destination table, segment both projections on the same column to avoid resegmenting the data, as in the following example:

```
CREATE TABLE source (col1 INT, col2 INT, col3 INT);
CREATE PROJECTION source_p (col1, col2, col3) AS
  SELECT col1, col2, col3 FROM source
  SEGMENTED BY HASH(col3) ALL NODES;
CREATE TABLE destination (col1 INT, col2 INT, col3 INT);
CREATE PROJECTION destination_p (col1, col2, col3) AS
  SELECT col1, col2, col3 FROM destination
  SEGMENTED BY HASH(col3) ALL NODES;
INSERT /*+direct*/ INTO destination SELECT * FROM source;
```

DELETE and UPDATE Queries

Vertica is optimized for query-intensive workloads, so DELETE and UPDATE queries might not achieve the same level of performance as other queries. A DELETE and UPDATE operation has to update all projections, so the operation is as slow as the slowest projection. For additional information, see [Using INSERT, UPDATE, and DELETE](#).

The topics that follow discuss best practices for optimizing DELETE and UPDATE queries in Vertica.

DELETE and UPDATE Performance Considerations

To improve the performance of your DELETE and UPDATE queries, consider the following issues:

- **Query performance after large deletes**—A large number of (unpurged) deleted rows can negatively affect query performance.

To eliminate rows that have been deleted from the result, a query must do extra processing. If 10% or more of the total rows in a table have been deleted, the performance of a query on the table degrades. However, your experience may vary depending on the size of the table, the table definition, and the query. If a table has a large number of deleted rows, consider purging those rows to improve performance. For more information on purging, see [Purging Deleted Data](#).

- **Recovery performance**—Recovery is the action required for a cluster to restore K-safety after a crash. Large numbers of deleted records can degrade the performance of a recovery. To improve recovery performance, purge the deleted rows. For more information on purging, see [Purging Deleted Data](#).
- **Concurrency**—DELETE and UPDATE take exclusive locks on the table. Only one DELETE or UPDATE transaction on a table can be in progress at a time and only when no loads (or INSERTs) are in progress. DELETES and UPDATES on different tables can be run concurrently.

For detailed tips about improving DELETE and UPDATE performance, see [DELETE and UPDATE Optimization](#).

Caution: Vertica does not remove deleted data immediately but keeps it as history for the purposes of historical query. A large amount of history can result in slower query performance. For information about how to configure the appropriate amount of history to retain, see [Purging Deleted Data](#).

DELETE and UPDATE Optimization

The process of optimizing DELETE and UPDATE queries is the same for both operations. Some simple steps can increase the query performance by tens to hundreds of times. The following sections describe several ways to improve projection design and improve DELETE and UPDATE queries to significantly increase DELETE and UPDATE performance.

Note: For large bulk deletion, Vertica recommends using [Partitioned Tables](#) where possible because they provide the best DELETE performance and improve query performance.

Projection Column Requirements for Optimized Deletes

When all columns required by the DELETE or UPDATE predicate are present in a projection, the projection is optimized for DELETES and UPDATES. DELETE and UPDATE operations on such projections are significantly faster than on non-optimized projections.

For example, consider the following table and projections:

```
CREATE TABLE t (a INTEGER, b INTEGER, c INTEGER);  
CREATE PROJECTION p1 (a, b, c) AS SELECT * FROM t ORDER BY a;  
CREATE PROJECTION p2 (a, c) AS SELECT a, c FROM t ORDER BY c, a;
```

In the following query, both p1 and p2 are eligible for DELETE and UPDATE optimization because column a is available:

```
DELETE from t WHERE a = 1;
```

In the following example, only projection p1 is eligible for DELETE and UPDATE optimization because the b column is not available in p2:

```
DELETE from t WHERE b = 1;
```

Optimized Deletes in Subqueries

To be eligible for DELETE optimization, all target table columns referenced in a DELETE or UPDATE statement's WHERE clause must be in the projection definition.

For example, the following simple schema has two tables and three projections:

```
CREATE TABLE tb1 (a INT, b INT, c INT, d INT);  
CREATE TABLE tb2 (g INT, h INT, i INT, j INT);
```

The first projection references all columns in `tb1` and sorts on column `a`:

```
CREATE PROJECTION tb1_p AS SELECT a, b, c, d FROM tb1 ORDER BY a;
```

The buddy projection references and sorts on column `a` in `tb1`:

```
CREATE PROJECTION tb1_p_2 AS SELECT a FROM tb1 ORDER BY a;
```

This projection references all columns in `tb2` and sorts on column `i`:

```
CREATE PROJECTION tb2_p AS SELECT g, h, i, j FROM tb2 ORDER BY i;
```

Consider the following DML statement, which references `tb1.a` in its `WHERE` clause. Since both projections on `tb1` contain column `a`, both are eligible for the optimized DELETE:

```
DELETE FROM tb1 WHERE tb1.a IN (SELECT tb2.i FROM tb2);
```

Restrictions

Optimized DELETES are not supported under the following conditions:

- With replicated projections if subqueries reference the target table. For example, the following syntax is not supported:

```
DELETE FROM tb1 WHERE tb1.a IN (SELECT e FROM tb2, tb2 WHERE tb2.e = tb1.e);
```

- With subqueries that do not return multiple rows. For example, the following syntax is not supported:

```
DELETE FROM tb1 WHERE tb1.a = (SELECT k from tb2);
```

Projection Sort Order for Optimizing Deletes

Design your projections so that frequently-used DELETE or UPDATE predicate columns appear in the sort order of all projections for large DELETES and UPDATES.

For example, suppose most of the DELETE queries you perform on a projection look like the following:

```
DELETE from t where time_key < '1-1-2007'
```

To optimize the DELETES, make `time_key` appear in the ORDER BY clause of all your projections. This schema design results in better performance of the DELETE operation.

In addition, add additional sort columns to the sort order such that each combination of the sort key values uniquely identifies a row or a small set of rows. For more information, see [Combine RLE and Sort Order](#). To analyze projections for sort order issues, use the [EVALUATE_DELETE_PERFORMANCE](#) function.

Data Collector Table Queries

The Vertica Data Collector extends system table functionality by gathering and retaining information about your database cluster. The Data Collector makes this information available in system tables.

Vertica Analytics Platform stores Data Collection data in the Data Collector directory under the Vertica or catalog path. Use Data Collector information to query the past state of system tables and extract aggregate information.

In general, queries on Data Collector tables are more efficient when they include only the columns that contain the desired data. Queries are also more efficient when they:

- [Avoid resegmentation](#)
- [Use time predicates](#)

Avoiding Resegmentation

You can avoid resegmentation when you join the following DC tables on `session_id` or `transaction_id`, because all data is local:

- `dc_session_starts`
- `dc_session_ends`
- `dc_requests_issued`
- `dc_requests_completed`

Resegmentation is not required when a query includes the `node_name` column. For example:

```
=> SELECT dri.transaction_id, dri.request, drc.processed_row_count
      FROM dc_requests_issued dri
      JOIN dc_requests_completed drc
      USING (node_name, session_id, request_id)
      WHERE dri.time between 'April 7,2015'::timestamptz and 'April 8,2015'::timestamptz
      AND drc.time between 'April 7,2015'::timestamptz and 'April 8,2015'::timestamptz;
```

This query runs efficiently because:

- The [initiator node](#) writes only to `dc_requests_issued` and `dc_requests_completed`.
- Columns `session_id` and `node_name` are correlated.

Using Time Predicates

Use non-volatile functions and [TIMESTAMP](#) for the time range predicates. Vertica Analytics Platform optimizes SQL performance for DC tables that use the time predicate.

Each DC table has a `time` column. Use this column to enter the time range as the query predicate.

For example, this query returns data for dates between September 1 and September 10:

```
select * from dc_foo where time > 'Sept 1, 2015::timestamptz and time < 'Sept 10
2015':: timestamptz;
```

You can change the minimum and maximum time values to adjust the time for which you want to retrieve data.

You must use non-volatile functions as time predicates. [Volatile functions](#) cause queries to run inefficiently. This example returns all queries that started and ended on April 7, 2015.

However, the query runs at less than optimal performance because `trunc` and `timestamp` are volatile:

```
=> SELECT dri.transaction_id, dri.request, drc.processed_row_count
      FROM dc_requests_issued dri
      LEFT JOIN dc_requests_completed drc
      USING (session_id, request_id)
      WHERE trunc(dri.time, 'DDD') > 'April 7,2015'::timestamp
      AND trunc(drc.time, 'DDD') < 'April 8,2015'::timestamp;
```

Views

A view is a stored query that encapsulates one or more `SELECT` statements. Views dynamically access and compute data from the database at execution time. A view is read-only, and can reference any combination of tables, temporary tables, and other views.

You can use views to achieve the following goals:

- Hide the complexity of `SELECT` statements from users for support or security purposes. For example, you can create a view that exposes only the data users need from various tables, while withholding sensitive data from the same tables.
- Encapsulate details about table structures, which might change over time, behind a consistent user interface.

Unlike projections, views are not materialized—that is, they do not store data on disk. Thus, the following restrictions apply:

- Vertica does not need to refresh view data when the underlying table data changes. However, a view does incur overhead to access and compute data.
- Views do not support inserts, deletes, or updates.

Creating Views

You can create two types of views:

- `CREATE VIEW` creates a view that persists across all sessions until it is explicitly dropped with `DROP VIEW`
- `CREATE LOCAL TEMPORARY VIEW` creates a view that is accessible only during the current Vertica session, and only to its creator. The view is automatically dropped when the current session ends.

After you create a view, you cannot change its definition. You can replace it with another view of the same name; or you can delete and redefine it.

Create Permissions

To create a view, you must be a superuser or have the following privileges:

Privilege	Objects
CREATE	Schema where the view is created.
SELECT	Tables and views referenced by the view query.
USAGE	All schemas that contain tables and views referenced by the view query.

For information about enabling users to access views, see [Enabling View Access](#).

Using Views

Views can be used in the FROM clause of any SQL query or subquery. At execution, Vertica internally substitutes the name of the view used in the query with the actual query used in the view definition.

CREATE VIEW Example

The following [CREATE VIEW](#) statement creates the view `myview`, which sums all individual incomes of customers listed in the `store.store_sales_fact` table, and groups results by state:

```
=> CREATE VIEW myview AS
  SELECT SUM(annual_income), customer_state FROM public.customer_dimension
  WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact)
  GROUP BY customer_state
  ORDER BY customer_state ASC;
```

You can use this view to find all combined salaries greater than \$2 billion:

```
=> SELECT * FROM myview where sum > 2000000000 ORDER BY sum DESC;
  SUM      | customer_state
-----+-----
29253817091 | CA
14215397659 | TX
5225333668  | MI
4907216137  | CO
4581840709  | IL
3769455689  | CT
3330524215  | FL
3310667307  | IN
2832710696  | TN
2806150503  | PA
2793284639  | MA
27234441590 | AZ
2642551509  | UT
```

```
2128169759 | NV  
(14 rows)
```

Enabling View Access

You can query any view that you create. To enable other non-superusers to access a view, you must grant them USAGE privileges to the view schema, and SELECT privileges to the view itself. The following example grants user2 access to view `schema1.view1`:

```
=> GRANT USAGE ON schema schema1 TO user2;  
=> GRANT SELECT ON schema1.view1 TO user2;
```

Important: If the view references an external table, you must also grant USAGE privileges to the external table's schema. So, if `schema1.view1` references external table `schema2.extTable1`, you must also grant user2 USAGE privileges to `schema2`:

```
=> GRANT USAGE on schema schema2 to user2;
```

For more information see [GRANT \(View\)](#).

View Execution

When Vertica processes a query that contains a view, it treats the view as a subquery. Vertica executes the query by expanding it to include the query in the view definition. For example, Vertica expands the query on the view `myview` shown in [CREATE VIEW Example](#), to include the query that the view encapsulates, as follows:

```
=> SELECT * FROM  
  (SELECT SUM(annual_income), customer_state FROM public.customer_dimension  
   WHERE customer_key IN  
     (SELECT customer_key FROM store.store_sales_fact)  
   GROUP BY customer_state  
   ORDER BY customer_state ASC)  
 AS ship where sum > 2000000000;
```

View Optimization

If you query a view and your query only includes columns from a subset of the tables that are joined in that view, Vertica executes that query by expanding it to include only those tables. This optimization requires one of the following conditions to be true:

- Join columns are foreign and primary keys.
- The join is a left or right outer join on columns with unique values.

View Sort Order

When processing a query on a view, Vertica considers the `ORDER BY` clause only in the outermost query. If the view definition includes an `ORDER BY` clause, Vertica ignores it. Thus, in order to sort the results returned by a view, you must specify the `ORDER BY` clause in the outermost query:

```
=> SELECT * FROM view-name ORDER BY view-column;
```

Note: One exception applies: Vertica sorts view data when the view includes a `LIMIT` clause. In this case, Vertica must sort the data before it can process the `LIMIT` clause.

For example, the following view definition contains an `ORDER BY` clause inside a `FROM` subquery:

```
=> CREATE VIEW myview AS SELECT SUM(annual_income), customer_state FROM public.customer_dimension
WHERE customer_key IN
  (SELECT customer_key FROM store.store_sales_fact)
GROUP BY customer_state
ORDER BY customer_state ASC;
```

When you query the view, Vertica does not sort the data:

```
=> SELECT * FROM myview WHERE SUM > 200000000;
  SUM      | customer_state
-----+-----
 5225333668 | MI
 2832710696 | TN
14215397659 | TX
 4907216137 | CO
 2793284639 | MA
 3769455689 | CT
 3310667307 | IN
 2723441590 | AZ
 2642551509 | UT
 3330524215 | FL
 2128169759 | NV
29253817091 | CA
 4581840709 | IL
 2806150503 | PA
(14 rows)
```

To return sorted results, the outer query must include an `ORDER BY` clause:

```
=> SELECT * FROM myview WHERE SUM > 200000000 ORDER BY customer_state ASC;
      SUM      | customer_state
-----+-----
  2723441590 | AZ
  29253817091 | CA
  4907216137 | CO
  3769455689 | CT
  3330524215 | FL
  4581840709 | IL
  3310667307 | IN
  2793284639 | MA
  5225333668 | MI
  2128169759 | NV
  2806150503 | PA
  2832710696 | TN
  14215397659 | TX
  2642551509 | UT
(14 rows)
```

Run-Time Errors

If Vertica does not have to evaluate an expression that would generate a run-time error in order to answer a query, the run-time error might not occur.

For example, the following query returns an error, because `TO_DATE` cannot convert the string `F` to the specified date format:

```
=> SELECT TO_DATE('F','dd mm yyyy') FROM customer_dimension;
ERROR: Invalid input for DD: "F"
```

Now create a view using the same query:

```
=> CREATE VIEW temp AS SELECT TO_DATE('F','dd mm yyyy')
      FROM customer_dimension;
CREATE VIEW
```

In many cases, this view generates the same error message. For example:

```
=> SELECT * FROM temp;
ERROR: Invalid input for DD: "F"
```

However, if you query that view with the `COUNT` function, Vertica returns with the desired results:

```
=> SELECT COUNT(*) FROM temp;
COUNT
-----
    100
(1 row)
```

This behavior works as intended. You can create views that contain subqueries, where not every row is intended to pass the predicate.

Managing Views

Obtaining View Information

You can query system tables [VIEWS](#) and [VIEW_COLUMNS](#) to obtain information about existing views—for example, a view's definition and the attributes of columns that comprise that view.

Renaming a View

Use [ALTER VIEW](#) to rename a view.

Dropping a View

Use [DROP VIEW](#) to drop a view. Only the specified view is dropped. Vertica does not support CASCADE functionality for views, and does not check for dependencies. Dropping a view causes any view that references it to fail.

Disabling and Re-enabling Views

If you drop a table that is referenced by a view, Vertica does not drop the view. However, attempts to use that view or access information about it from system table [VIEW_COLUMNS](#) return an error that the referenced table does not exist. If you restore that table, Vertica also re-enables usage of the view.

Flattened Tables

Highly normalized database design often uses a star or snowflake schema model, comprising multiple large fact tables and many smaller dimension tables. Queries typically involve joins between a large fact table and multiple dimension tables. Depending on the number of tables and quantity of data that are joined, these queries can incur significant overhead.

To avoid this problem, some users create wide tables that combine all fact and dimension table columns that their queries require. These tables can dramatically speed up query execution. However, maintaining redundant sets of normalized and denormalized data has its own administrative costs.

Denormalized, or *flattened*, tables, can minimize these problems. Flattened tables can include columns that get their values by querying other tables. Operations on the source tables and flattened table are decoupled; changes in one are not automatically propagated to the other. This minimizes the overhead that is otherwise typical of denormalized tables.

A flattened table defines derived columns with one of the following [column constraint](#) clauses:

- `DEFAULT query-expression` sets the column value on two events:
 - When the column is created with `CREATE TABLE` or `ALTER TABLE . . . ADD COLUMN`.
 - Any table load operation such as `INSERT`.
- `SET USING query-expression` sets the column value only when the function `REFRESH_COLUMNS` is invoked.

In both cases, *query-expression* must return only one row and column value, or none. If the query returns no rows, the column value is set to `NULL`.

Like other tables defined in Vertica, you can add and remove `DEFAULT` and `SET USING` columns from a flattened table at any time. Vertica enforces dependencies between a flattened table and the tables that it queries. For details, see [Modifying SET USING and DEFAULT Columns](#).

Flattened Table Example

In the following example, columns `orderFact.cust_name` and `orderFact.cust_gender` are defined as `SET USING` and `DEFAULT` columns, respectively. Both columns obtain their values by querying table `custDim`:

```
=> CREATE TABLE public.custDim(  
    cid int PRIMARY KEY NOT NULL,  
    name varchar(20),  
    age int,  
    gender varchar(1)  
);  
  
=> CREATE TABLE public.orderFact(  
    order_id int PRIMARY KEY NOT NULL,  
    cid int REFERENCES public.custDim(cid),  
    cust_name varchar(20) SET USING (SELECT name FROM public.custDim WHERE (custDim.cid =  
orderFact.cid)),  
    cust_gender varchar(1) DEFAULT (SELECT gender FROM public.custDim WHERE (custDim.cid =  
orderFact.cid)),  
    amount numeric(12,2)  
);
```

The following INSERT commands load data into both tables:

```
=> INSERT INTO custDim VALUES(1, 'Alice', 25, 'F');  
=> INSERT INTO custDim VALUES(2, 'Boz', 30, 'M');  
=> INSERT INTO custDim VALUES(3, 'Eva', 32, 'F');  
=>  
=> INSERT INTO orderFact (order_id, cid, amount) VALUES(100, 1, 15);  
=> INSERT INTO orderFact (order_id, cid, amount) VALUES(200, 1, 1000);  
=> INSERT INTO orderFact (order_id, cid, amount) VALUES(300, 2, -50);  
=> INSERT INTO orderFact (order_id, cid, amount) VALUES(400, 3, 100);  
=> INSERT INTO orderFact (order_id, cid, amount) VALUES(500, 2, 200);  
=> COMMIT;
```

When you query the tables, Vertica returns the following result sets:

```
=> SELECT * FROM custDim;  
cid | name | age | gender  
-----+-----+-----+-----  
  1 | Alice |  25 | F  
  2 | Boz  |  30 | M  
  3 | Eva  |  32 | F  
(3 rows)  
  
=> SELECT * FROM orderFact ORDER BY cid;  
order_id | cid | cust_name | cust_gender | amount  
-----+-----+-----+-----+-----  
   100 |  1 |           | F           |  15.00  
   200 |  1 |           | F           | 1000.00  
   300 |  2 |           | M           |  -50.00  
   500 |  2 |           | M           |  200.00  
   400 |  3 |           | F           |  100.00  
(5 rows)
```

Vertica automatically populates the DEFAULT column `orderFact.cust_gender`, but the SET USING column `orderFact.cust_name` remains NULL. To populate this column, you must call the function [REFRESH_COLUMNS](#) on flattened table `orderFact`. This function invokes the SET USING query for column `orderFact.cust_name` and populates the column from the result set:

```
=> SELECT REFRESH_COLUMNS('orderFact', 'cust_name', 'REBUILD');

      REFRESH_COLUMNS
-----
refresh_columns completed
(1 row)

=> COMMIT;
COMMIT
=> SELECT * FROM orderFact ORDER BY cid;
 order_id | cid | cust_name | cust_gender | amount
-----+-----+-----+-----+-----
      100 |   1 | Alice     | F           |  15.00
      200 |   1 | Alice     | F           | 1000.00
      300 |   2 | Boz       | M           |  -50.00
      500 |   2 | Boz       | M           |  200.00
      400 |   3 | Eva      | F           |  100.00
(5 rows)
```

Creating Flattened Tables

A flattened table is typically a fact table where one or more columns query other tables for their values, through `DEFAULT` or `SET USING` constraints. Like other columns, you can set these constraints when you create the flattened table, or any time thereafter by modifying the table schema:

```
CREATE TABLE...(..., column-name data-type { DEFAULT | SET USING } expression,...)
```

```
ALTER TABLE...ADD COLUMN column-name { DEFAULT | SET USING } expression
```

```
ALTER TABLE...ALTER COLUMN column-name { SET DEFAULT | SET USING } expression
```

`DEFAULT` and `SET USING` constraints can be used for columns of all data types. The expressions that you set for these constraints are stored in the system table [COLUMNS](#), in columns `COLUMN_DEFAULT` and `COLUMN_SET_USING`. Both constraints support the same expressions. For detailed information, including restrictions, see [Defining Column Values](#) in the Administrator's Guide.

Supported Expressions

`DEFAULT` and `SET USING` generally support the same expressions. These include:

- Queries (see [Flattened Tables](#))
- Other columns in the same table
- [Literals](#) (constants)

- All [operators](#) supported by Vertica
- The following categories of functions:
 - [Null-handling](#)
 - [User-defined scalar](#)
 - [System information](#)
 - [String](#)
 - [Mathematical](#)
 - [Formatting](#)

Disambiguating Predicate Columns

If a `SET USING` or `DEFAULT` query expression joins two columns with the same name, the column names must include their table names. Otherwise, Vertica assumes that both columns reference the dimension table, and the predicate always evaluates to true.

For example, tables `orderFact` and `custDim` both include column `cid`. Flattened table `orderFact` defines column `cust_name` with a `SET USING` query expression. Because the query predicate references columns `cid` from both tables, the column names are fully qualified:

```
CREATE TABLE public.orderFact
(
  ...
  cid int REFERENCES public.custDim(cid),
  cust_name varchar(20) SET USING (
    SELECT name FROM public.custDim WHERE (custDIM.cid = orderFact.cid)),
  ...
)
```

Required Privileges

Flattened tables require the following privileges:

- To view flattened table data: `READ` privileges. No privileges are required on queried tables.
- To add `SET USING` or `DEFAULT` columns that query other tables: `SELECT` privileges on the queried tables, `CREATE` privileges on the flattened table.
- To load data: `SELECT` privileges on queried tables.

Running `REFRESH_COLUMNS` requires the following privileges on the flattened table:

- MODIFY privilege on the table, USAGE privilege on its schema
- For each SET USING column that queries another table or view: SELECT privilege on the queried table, USAGE privilege on its schema.

SET USING versus DEFAULT

Columns in a flattened table can query other tables with constraints SET USING and DEFAULT. In both cases, changes in the queried tables are not automatically propagated to the flattened table. The two constraints differ as described below.

DEFAULT Columns

Vertica executes DEFAULT queries only on new rows when they are added to the flattened table, through load operations such as INSERT and COPY. Thereafter, changes in the original data sources have no effect on the flattened table.

SET USING Columns

Vertica executes SET USING queries only when you invoke the function [REFRESH_COLUMNS](#). Load operations set SET USING columns in new rows to NULL. After the load, you must call REFRESH_COLUMNS to populate these columns from the queried tables. This can be useful in two ways: you can defer the overhead of updating the flattened table to any time that is convenient; and you can repeatedly query source tables for new data.

One exception applies: when you use [ALTER TABLE . . . ALTER COLUMN](#) to apply SET USING to an existing column, or modify an existing SET USING expression. In this case, the DDL operation automatically invokes REFRESH_COLUMNS on the column, using UPDATE mode. After the refresh operation is complete, the DDL operation auto-commits the updates and returns. If the refresh operation fails, Vertica rolls back the entire DDL operation. Execution time can be significant if the refresh operation involves a large data set.

Tip: It might be more efficient to drop the column and add a new one.

[ALTER TABLE . . . ADD COLUMN](#) does not call REFRESH_COLUMNS when you add a SET USING column to a table.

SET USING is especially useful for large flattened tables that reference data from multiple dimension tables. Often, only a small subset of SET USING columns are subject to change, and

queries on the flattened table do not always require up-to-the-minute data. Given this scenario, you can refresh table content at regular intervals, or only during off-peak hours. One or both of these strategies can minimize maintenance costs, and facilitate performance when querying large data sets.

Combining DEFAULT and SET USING Constraints

A column can specify both DEFAULT and SET USING constraints, as follows:

```
column-name data-type DEFAULT default-expr SET USING using-expr
```

Typically, both constraints specify the same expression. In this case, you can define the column as follows:

```
column-name data-type DEFAULT USING expression
```

DEFAULT USING columns support the same expressions as SET USING columns, and are subject to the same [restrictions](#).

Example

The following SQL illustrates differences between SET USING and DEFAULT constraints. The examples use the custDim and orderFact tables described in [Flattened Table Example](#).

The following [UPDATE](#) statement updates the custDim table:

```
=> UPDATE custDim SET name='Roz', gender='F' WHERE cid=2;
OUTPUT
-----
      1
(1 row)

=> COMMIT;
COMMIT
```

Changes are not propagated to flattened table orderFact, which includes SET USING and DEFAULT columns cust_name and cust_gender, respectively:

```
=> SELECT * FROM custDim ORDER BY cid;
cid | name | age | gender
-----+-----+-----+-----
  1 | Alice | 25 | F
  2 | Roz  | 30 | F
  3 | Eva  | 32 | F
(3 rows)

=> SELECT * FROM orderFact ORDER BY cid;
order_id | cid | cust_name | cust_gender | amount
-----+-----+-----+-----+-----
```

```
100 | 1 | Alice | F | 15.00
200 | 1 | Alice | F | 1000.00
300 | 2 | Boz | M | -50.00
500 | 2 | Boz | M | 200.00
400 | 3 | Eva | F | 100.00
(5 rows)
```

The following INSERT statement invokes the cust_gender column's DEFAULT query and sets that column to F. The load operation does not invoke the cust_name column's SET USING query, so cust_name is set to null:

```
=> INSERT INTO orderFact(order_id, cid, amount) VALUES(500, 3, 750);
OUTPUT
-----
1
(1 row)

=> COMMIT;
COMMIT
=> SELECT * FROM orderFact ORDER BY cid;
 order_id | cid | cust_name | cust_gender | amount
-----+-----+-----+-----+-----
100 | 1 | Alice | F | 15.00
200 | 1 | Alice | F | 1000.00
300 | 2 | Boz | M | -50.00
500 | 2 | Boz | M | 200.00
400 | 3 | Eva | F | 100.00
500 | 3 | | F | 750.00
(6 rows)
```

To set a value in cust_name, invoke its SET USING query by calling REFRESH_COLUMNS:

```
=> SELECT REFRESH_COLUMNS ('orderFact','');
REFRESH_COLUMNS
-----
refresh_columns completed
(1 row)

=> COMMIT;
COMMIT
=> SELECT * from orderFact ORDER BY cid;
 order_id | cid | cust_name | cust_gender | amount
-----+-----+-----+-----+-----
100 | 1 | Alice | F | 150.00
200 | 1 | Alice | F | 1000.00
300 | 2 | Roz | M | -50.00
500 | 2 | Roz | M | 200.00
400 | 3 | Eva | F | 100.00
500 | 3 | Eva | F | 750.00
(6 rows)
```

REFRESH_COLUMNS executes cust_name's SET USING query: it queries the name column in table custDim and updates cust_name with the following values:

- Sets `cust_name` in the new row to Eva.
- Returns updated values for `cid=2`, and changes Boz to Roz.

`REFRESH_COLUMNS` only affects the values in column `cust_name`. Values in column `gender` are unchanged, so settings for rows where `cid=2` (Roz) remain set to M. To repopulate `orderFact.cust_gender` with default values from `custDim.gender`, call `UPDATE` on `orderFact`:

```
=> UPDATE orderFact SET cust_gender=DEFAULT WHERE cust_name='Roz';
OUTPUT
-----
      2
(1 row)

=> COMMIT;
COMMIT
=> SELECT * FROM orderFact ORDER BY cid;
 order_id | cid | cust_name | cust_gender | amount
-----+-----+-----+-----+-----
    100 |  1 | Alice     | F           |  15.00
    200 |  1 | Alice     | F           | 1000.00
    300 |  2 | Roz       | F           |  -50.00
    500 |  2 | Roz       | F           |  200.00
    400 |  3 | Eva       | F           |  100.00
    500 |  3 | Eva       | F           |  750.00
(6 rows)
```

Modifying SET USING and DEFAULT Columns

You can drop `SET USING` and `DEFAULT` columns from a flattened table at any time. Doing so has no effect on the tables that these columns query.

Removing SET USING and DEFAULT Constraints

You remove a column's `SET USING` or `DEFAULT` constraint with `ALTER TABLE...ALTER COLUMN`, as follows:

```
ALTER TABLE table-name ALTER COLUMN column-name DROP { SET USING | DEFAULT };
```

Vertica removes the constraint from the specified column, but the column and its data are otherwise unaffected.

Modifying a SET USING and DEFAULT Expression

You can change the query expression of an existing SET USING or DEFAULT column by calling `ALTER TABLE...ALTER COLUMN`. When you modify a SET USING column expression, Vertica automatically invokes `REFRESH_COLUMNS` using UPDATE mode, and repopulates that column. If a large number of rows must be refreshed, the following approach might be preferable:

1. Drop the column with `ALTER TABLE...DROP COLUMN`.
2. Call `ALTER TABLE...ADD COLUMN` to add a new one with the desired SET USING expression.

This approach has two benefits:

- Avoids the immediate call to `REFRESH_COLUMNS`, and defers the overhead that is otherwise incurred.
- Lets you specify `REBUILD` mode when you call `REFRESH_COLUMNS`, which in many cases repopulates the column more efficiently.

Dropping Columns Queried by SET USING or DEFAULT

Vertica enforces dependencies between a flattened table and the tables that it queries. Attempts to drop a queried column or its table return an error unless the drop operation, such as `DROP TABLE`, also includes `CASCADE`. Vertica implements `CASCADE` by removing the SET USING or DEFAULT constraint from the flattened table. The table column and its data are otherwise unaffected.

For example, attempts to drop column `name` in table `custDim` returns a rollback error, as this column is referenced by SET USING column `cust_name` in `orderFact`:

```
=> ALTER TABLE custDim DROP COLUMN name;  
ROLLBACK 7302: Cannot drop column "name" since it is referenced in the set using expression of table  
"public.orderFact", column "cust_name"
```

To drop this column, use the `CASCADE` option:

```
=> ALTER TABLE custDim DROP COLUMN name CASCADE;  
ALTER TABLE
```

Vertica removes the `SET USING` constraint from `orderFact.cust_name` as part of the drop operation. However, `cust_name` retains the data that it derived from dropped column `custDim.name`:

```
=> SELECT EXPORT_TABLES('', 'orderFact');
                                EXPORT_TABLES
-----
CREATE TABLE public.orderFact
(
  order_id int NOT NULL,
  cid int,
  cust_name varchar(20)
  cust_gender varchar(1) DEFAULT ( SELECT custDim.gender
FROM public.custDim
WHERE (custDim.cid = orderFact.cid)),
  amount numeric(12,2)
);
ALTER TABLE public.orderFact ADD CONSTRAINT C_PRIMARY PRIMARY KEY (order_id) ENABLED;
ALTER TABLE public.orderFact ADD CONSTRAINT C_FOREIGN FOREIGN KEY (cid) references public.custDim
(cid);
(1 row)

=> SELECT * FROM orderFact;
order_id | cid | cust_name | cust_gender | amount
-----+-----+-----+-----+-----
    300 |  2 | Roz       | F           | -50.00
    400 |  3 | Eva       | F           | 100.00
    100 |  1 | Alice     | F           |  15.00
    500 |  2 | Roz       | F           | 200.00
    200 |  1 | Alice     | F           | 1000.00
(5 rows)
```

Impact of SET USING Columns on License Limits

Vertica does not count the data in denormalized columns towards your raw data license limit. `SET USING` columns obtain their data by querying columns in other tables. You cannot modify `SET USING` data directly—for example, through DML operations such as `INSERT` and `COPY`. Thus, only the source data counts against your raw data license limit.

For a list of `SET USING` restrictions, see [Defining Column Values](#).

SQL Analytics

Vertica analytics are SQL functions based on the ANSI 99 standard. These functions handle complex analysis and reporting tasks—for example:

- Rank the longest-standing customers in a particular state.
- Calculate the moving average of retail volume over a specified time.
- Find the highest score among all students in the same grade.
- Compare the current sales bonus that salespersons received against their previous bonus.

Analytic functions return aggregate results but they do not group the result set. They return the group value multiple times, once per record. You can sort group values, or partitions, using a window `ORDER BY` clause, but the order affects only the function result set, not the entire query result set.

For details about supported functions, see [Analytic Functions](#) in the SQL Reference Manual.

Invoking Analytic Functions

You invoke analytic functions as follows:

```
analytic-function (arguments) OVER(  
    [ window-partition-clause ]  
    [ window-order-clause [ window-frame-clause ] ]  
)
```

An analytic function's `OVER` clause can contain up to three sub-clauses, which specify how to partition and sort function input, and how to frame input with respect to the current row. Function input is the result set that the query returns after it evaluates `FROM`, `WHERE`, `GROUP BY`, and `HAVING` clauses.

Each function has its own `OVER` clause requirements. For example, some analytic functions do not support window order and window frame clauses.

Function Execution

An analytic function executes as follows:

1. Takes the input rows that the query returns after it performs all joins, and evaluates FROM, WHERE, GROUP BY, and HAVING clauses.
2. Groups input rows according to the window partition (PARTITION BY) clause. If this clause is omitted, all input rows are treated as a single partition.
3. Sorts rows within each partition according to window order (ORDER BY) clause.
4. If the OVER clause includes a window order clause, the function checks for a window frame clause and executes it as it processes each input row. If the OVER clause omits a window frame clause, the function treats the entire partition as a window frame.

Restrictions

- Analytic functions are allowed only in a query's SELECT and ORDER BY clauses.
- Analytic functions cannot be nested. For example, the following query throws an error:

```
=> SELECT MEDIAN(RANK() OVER(ORDER BY sal) OVER()).
```

Analytic Functions Versus Aggregate Functions

Like aggregate functions, analytic functions return aggregate results, but analytics do not group the result set. Instead, they return the group value multiple times with each record, allowing further analysis.

Analytic queries generally run faster and use fewer resources than aggregate queries.

Aggregate functions	Analytic functions
Return a single summary value.	Return the same number of rows as the input.
Define the groups of rows on which they operate through the SQL GROUP BY clause.	Define the groups of rows on which they operate through window partition and window frame clauses.

Examples

The examples below contrast the aggregate function `COUNT` with its analytic counterpart `COUNT`. The examples use the `employees` table as defined below:

```
CREATE TABLE employees(emp_no INT, dept_no INT);
INSERT INTO employees VALUES(1, 10);
INSERT INTO employees VALUES(2, 30);
INSERT INTO employees VALUES(3, 30);
INSERT INTO employees VALUES(4, 10);
INSERT INTO employees VALUES(5, 30);
INSERT INTO employees VALUES(6, 20);
INSERT INTO employees VALUES(7, 20);
INSERT INTO employees VALUES(8, 20);
INSERT INTO employees VALUES(9, 20);
INSERT INTO employees VALUES(10, 20);
INSERT INTO employees VALUES(11, 20);
COMMIT;
```

When you query this table, the following result set returns:

```
=> SELECT * FROM employees ORDER BY emp_no;
 emp_no | dept_no
-----+-----
      1 |     10
      2 |     30
      3 |     30
      4 |     10
      5 |     30
      6 |     20
      7 |     20
      8 |     20
      9 |     20
     10 |     20
     11 |     20
(11 rows)
```

Below, two queries use the `COUNT` function to count the number of employees in each department. The query on the left uses aggregate function `COUNT`; the query on the right uses analytic function `COUNT`:

Aggregate COUNT	Analytics COUNT
<pre>=> SELECT dept_no, COUNT(*) AS emp_count FROM employees GROUP BY dept_no ORDER BY dept_no;</pre>	<pre>=> SELECT emp_no, dept_no, COUNT(*) OVER(PARTITION BY dept_no ORDER BY emp_no) AS emp_count FROM employees;</pre>
<pre>dept_no emp_count -----+-----</pre>	<pre>emp_no dept_no emp_count</pre>

<pre>10 2 20 6 30 3 (3 rows)</pre>	<pre>-----+-----+----- 1 10 1 4 10 2 -----+-----+----- 6 20 1 7 20 2 8 20 3 9 20 4 10 20 5 11 20 6 -----+-----+----- 2 30 1 3 30 2 5 30 3 (11 rows)</pre>
<p>Aggregate function COUNT returns one row per department for the number of employees in that department.</p>	<p>Within each dept_no partition analytic function COUNT returns a cumulative count of employees. The count is ordered by emp_no, as specified by the ORDER BY clause.</p>

See Also

- [Analytic Query Examples](#)

Window Partitioning

Optionally specified in an analytic function's OVER clause, a partition (PARTITION BY) clause groups input rows before the function processes them. Window partitioning is similar to an aggregate function's GROUP BY clause, except it returns exactly one result row per input row. If you omit the window partition clause, the function treats all input rows as a single partition.

Specifying Window Partitioning

You specify window partitioning in the analytic function's OVER clause, as shown below. The window partition clause is set off in bold:

```
OVER( { PARTITION BY expression | PARTITION BEST | PARTITION NODES }
      window-order-clause
      window-frame-clause )
```

For syntax details, see [Window Partition Clause](#) in the SQL Reference Manual.

Examples

The examples in this topic use the allsales schema defined in [Invoking Analytic Functions](#).

```
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT);
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

Median of sales within each state

The following query uses the analytic *window-partition-clause* to calculate the median of sales within each state. The analytic function is computed per partition and starts over again at the beginning of the next partition.

```
=> SELECT state, name, sales, MEDIAN(sales)
      OVER (PARTITION BY state) AS median from allsales;
```

Results are grouped into partitions for MA (35) and NY (20) under the median column.

state	name	sales	median
NY	C	15	20
NY	B	20	20
NY	F	40	20
MA	G	10	35
MA	D	20	35
MA	E	50	35
MA	A	60	35

(7 rows)

Median of sales among all states

The following query calculates the median of total sales among states. When you use `OVER()` with no parameters, there is one partition—the entire input:

```
=> SELECT state, sum(sales), median(SUM(sales))
      OVER () AS median FROM allsales GROUP BY state;
```

state	sum	median
NY	75	107.5
MA	140	107.5

(2 rows)

Sales larger than median (evaluation order)

Analytic functions are evaluated after all other clauses except the query's final `SQL ORDER BY`

clause. So a query that asks for all rows with sales larger than the median returns an error because the WHERE clause is applied before the analytic function and column m does not yet exist:

```
=> SELECT name, sales, MEDIAN(sales) OVER () AS m
   FROM allsales WHERE sales > m;
ERROR 2624: Column "m" does not exist
```

You can work around this by placing in a subquery the predicate WHERE sales > m:

```
=> SELECT * FROM
   (SELECT name, sales, MEDIAN(sales) OVER () AS m FROM allsales) sq
  WHERE sales > m;
 name | sales | m
-----+-----+---
 F    |    40 | 20
 E    |    50 | 20
 A    |    60 | 20
(3 rows)
```

For more examples, see [Analytic Query Examples](#).

Window Ordering

Window ordering specifies how to sort rows that are supplied to the analytic function. You specify window ordering through an ORDER BY clause in the function's OVER clause, as shown below. If the OVER clause includes a [window partition clause](#), rows are sorted within each partition. An window order clause also creates a default [window frame](#) if none is explicitly specified.

The window order clause only specifies order within a window result set. The query can have its own [ORDER BY](#) clause outside the OVER clause. This has precedence over the window order clause and orders the final result set.

Specifying Window Order

You specify a window frame in the analytic function's OVER clause, as shown below. The window order clause is set off in bold:

```
OVER(
  [ window-partition-clause ]
  ORDER BY { expression [ sort-qualifiers ] } [,...]
  [ window-frame-clause ]
)
sort-qualifiers =
```

```
[ ASC | DESC ]  
[ NULLS { FIRST | LAST | AUTO} ]
```

For syntax details, see [Window Order Clause](#) in the SQL Reference Manual.

Analytic Function Usage

Analytic aggregation functions such as [SUM](#) support window order clauses.

Required Usage

The following functions require a window order clause:

- [RANK](#)
- [DENSE_RANK](#)
- [LEAD](#)
- [LAG](#)
- [PERCENT_RANK](#)
- [CUME_DIST](#)
- [NTILE](#)

Invalid Usage

You cannot use a window order clause with the following functions:

- [PERCENTILE_CONT](#)
- [PERCENTILE_DISC](#)
- [MEDIAN](#)

Examples

The examples below use the `allsales` table schema:

```
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT);  
INSERT INTO allsales VALUES('MA', 'A', 60);  
INSERT INTO allsales VALUES('NY', 'B', 20);  
INSERT INTO allsales VALUES('NY', 'C', 15);  
INSERT INTO allsales VALUES('MA', 'D', 20);  
INSERT INTO allsales VALUES('MA', 'E', 50);
```

```
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

Example 1

The following query orders sales inside each state partition:

```
=> SELECT state, sales, name, RANK() OVER(
  PARTITION BY state
  ORDER BY sales) AS RANK
FROM allsales;
```

state	sales	name	RANK
MA	10	G	1
MA	20	D	2
MA	50	E	3
MA	60	A	4
NY	15	C	1
NY	20	B	2
NY	40	F	3

(7 rows)

Example 2

The following query's final ORDER BY clause sorts results by name:

```
=> SELECT state, sales, name, RANK() OVER(
  PARTITION by state
  ORDER BY sales) AS RANK
FROM allsales ORDER BY name;
```

state	sales	name	RANK
MA	60	A	4
NY	20	B	2
NY	15	C	1
MA	20	D	2
MA	50	E	3
NY	40	F	3
MA	10	G	1

(7 rows)

Window Framing

The window frame of an analytic function comprises a set of rows relative to the row that is currently being evaluated by the function. After the analytic function processes that row and its window frame, Vertica advances the current row and adjusts the frame boundaries accordingly. If the `OVER` clause also specifies a [partition](#), Vertica also checks that frame boundaries do not cross partition boundaries. This process repeats until the function evaluates the last row of the last partition.

Specifying a Window Frame

You specify a window frame in the analytic function's `OVER` clause, as shown below. The window frame clause is set off in bold:

```
OVER(  
  [ window-partition-clause ]  
  window-order-clause  
  { ROWS | RANGE } { BETWEEN start-point AND end-point } | start-point  
)
```

start-point and *end-point* specify the window frame's offset from the current row, as follows:

```
{ UNBOUNDED {PRECEDING | FOLLOWING}  
  | CURRENT ROW  
  | constant-value {PRECEDING | FOLLOWING}  
}
```

The keywords `ROWS` and `RANGE` specify whether the offset is physical or logical. If you specify only a start point, Vertica creates a window from that point to the current row.

For syntax details, see [window-frame-clause](#) in the SQL Reference Manual.

Requirements

In order to specify a window frame, the `OVER` must also specify a [window order](#) (`ORDER BY`) clause. If the `OVER` clause includes a window order clause but omits specifying a window frame, the function creates a default frame that extends from the first row in the current partition to the current row. This is equivalent to the following clause:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Window Aggregate Functions

Analytic functions that support window frames are called window aggregates. They return information such as moving averages and cumulative results. To use the following functions as window (analytic) aggregates, instead of basic aggregates, the `OVER` clause must specify a [window order](#) clause and, optionally, a window frame clause. If the `OVER` clause omits specifying a window frame, the function creates a default window frame as described earlier.

The following analytic functions support window frames:

- [AVG](#)
- [COUNT](#)
- [MAX](#) / [MIN](#)
- [SUM](#)
- [STDDEV](#) / [STDDEV_POP](#) / [STDDEV_SAMP](#)
- [VARIANCE](#) / [VAR_POP](#) / [VAR_SAMP](#)

Note: Functions [FIRST_VALUE](#) and [LAST_VALUE](#) also support window frames, but they are only analytic functions with no aggregate counterpart. [EXPONENTIAL_MOVING_AVERAGE](#), [LAG](#), and [LEAD](#) analytic functions do not support window frames.

A window aggregate with an empty `OVER` clause creates no window frame. The function is used as a reporting function, where all input is treated as a single partition.

Windows with a Physical Offset (ROWS)

The keyword `ROWS` in a window frame clause specifies window dimensions as the number of rows relative to the current row. The value can be `INTEGER` data type only.

Note: The value returned by an analytic function with a physical offset is liable to produce nondeterministic results unless the ordering expression results in a unique ordering. To achieve unique ordering, the [window order clause](#) might need to specify multiple columns.

Examples

The examples on this page use the emp table schema:

```
CREATE TABLE emp(deptno INT, sal INT, empno INT);
INSERT INTO emp VALUES(10,101,1);
INSERT INTO emp VALUES(10,104,4);
INSERT INTO emp VALUES(20,100,11);
INSERT INTO emp VALUES(20,109,7);
INSERT INTO emp VALUES(20,109,6);
INSERT INTO emp VALUES(20,109,8);
INSERT INTO emp VALUES(20,110,10);
INSERT INTO emp VALUES(20,110,9);
INSERT INTO emp VALUES(30,102,2);
INSERT INTO emp VALUES(30,103,3);
INSERT INTO emp VALUES(30,105,5);
COMMIT;
```

The following query invokes COUNT to count the current row and the rows preceding it, up to two rows:

```
SELECT deptno, sal, empno, COUNT(*) OVER
      (PARTITION BY deptno ORDER BY sal ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
      AS count FROM emp;
```

The OVER clause contains three components:

- Window partition clause PARTITION BY deptno
- Order by clause ORDER BY sal
- Window frame clause ROWS BETWEEN 2 PRECEDING AND CURRENT ROW. This clause defines window dimensions as extending from the current row through the two rows that precede it.

The query returns results that are divided into three partitions, indicated below as red lines. Within the second partition (deptno=20), COUNT processes the window frame clause as follows:

1. Creates the first window (green box). This window comprises a single row, as the current row (blue box) is also the the partition's first row. Thus, the value in the count column shows the number of rows in the current window, which is 1:

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	
20	109	6	
20	109	8	
20	110	10	
20	110	9	
30	102	2	1
30	103	3	2
30	105	5	3

- After COUNT processes the partition's first row, it resets the current row to the partition's second row. The window now spans the current row and the row above it, so COUNT returns a value of 2:

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	2
20	109	6	
20	109	8	
20	110	10	
20	110	9	
30	102	2	1
30	103	3	2
30	105	5	3

- After COUNT processes the partition's second row, it resets the current row to the partition's third row. The window now spans the current row and the two rows above it, so

COUNT returns a value of 3:

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	2
20	109	6	3
20	109	8	
20	110	10	
20	110	9	
30	102	2	1
30	103	3	2
30	105	5	3

- Thereafter, COUNT continues to process the remaining partition rows and moves the window accordingly, but the window dimensions (ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) remain unchanged as three rows. Accordingly, the value in the count column also remains unchanged (3):

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	2
20	109	6	3
20	109	8	3
20	110	10	
20	110	9	
30	102	2	1
30	103	3	2
30	105	5	3

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	2
20	109	6	3
20	109	8	3
20	110	10	3
20	110	9	3
30	102	2	1
30	103	3	2
30	105	5	3

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	2
20	109	6	3
20	109	8	3
20	110	10	3
20	110	9	3
30	102	2	1
30	103	3	2
30	105	5	3

Windows with a Logical Offset (RANGE)

The RANGE keyword defines an analytic window frame as a logical offset from the current row.

Note: The value returned by an analytic function with a logical offset is always deterministic.

For each row, an analytic function uses the [window order clause](#) (ORDER_BY) column or expression to calculate window frame dimensions as follows:

1. Within the current partition, evaluates the ORDER_BY value of the current row against the ORDER_BY values of contiguous rows.
2. Determines which of these rows satisfy the specified range requirements relative to the current row.
3. Creates a window frame that includes only those rows.
4. Executes on the current window.

Example

This example uses the table `property_sales`, which contains data about neighborhood home sales:

```
=> SELECT property_key, neighborhood, sell_price FROM property_sales ORDER BY neighborhood, sell_price;
```

property_key	neighborhood	sell_price
10918	Jamaica Plain	353000
10921	Jamaica Plain	450000
10927	Jamaica Plain	450000
10922	Jamaica Plain	474000
10919	Jamaica Plain	515000
10917	Jamaica Plain	675000
10924	Jamaica Plain	675000
10920	Jamaica Plain	705000
10923	Jamaica Plain	710000
10926	Jamaica Plain	875000
10925	Jamaica Plain	900000
10930	Roslindale	300000
10928	Roslindale	422000
10932	Roslindale	450000
10929	Roslindale	485000
10931	Roslindale	519000
10938	West Roxbury	479000
10933	West Roxbury	550000
10937	West Roxbury	550000
10934	West Roxbury	574000
10935	West Roxbury	598000
10936	West Roxbury	615000
10939	West Roxbury	720000

(23 rows)

The analytic function `AVG` can obtain the average of proximate selling prices within each neighborhood. The following query calculates for each home the average sale for all other neighborhood homes whose selling price was \$50k higher or lower:

```
=> SELECT property_key, neighborhood, sell_price, AVG(sell_price) OVER(  
    PARTITION BY neighborhood ORDER BY sell_price  
    RANGE BETWEEN 50000 PRECEDING and 50000 FOLLOWING)::int AS comp_sales
```

```
FROM property_sales ORDER BY neighborhood;
property_key | neighborhood | sell_price | comp_sales
-----+-----+-----+-----
10918 | Jamaica Plain | 353000 | 353000
10927 | Jamaica Plain | 450000 | 458000
10921 | Jamaica Plain | 450000 | 458000
10922 | Jamaica Plain | 474000 | 472250
10919 | Jamaica Plain | 515000 | 494500
10917 | Jamaica Plain | 675000 | 691250
10924 | Jamaica Plain | 675000 | 691250
10920 | Jamaica Plain | 705000 | 691250
10923 | Jamaica Plain | 710000 | 691250
10926 | Jamaica Plain | 875000 | 887500
10925 | Jamaica Plain | 900000 | 887500
10930 | Roslindale | 300000 | 300000
10928 | Roslindale | 422000 | 436000
10932 | Roslindale | 450000 | 452333
10929 | Roslindale | 485000 | 484667
10931 | Roslindale | 519000 | 502000
10938 | West Roxbury | 479000 | 479000
10933 | West Roxbury | 550000 | 568000
10937 | West Roxbury | 550000 | 568000
10934 | West Roxbury | 574000 | 577400
10935 | West Roxbury | 598000 | 577400
10936 | West Roxbury | 615000 | 595667
10939 | West Roxbury | 720000 | 720000
(23 rows)
```

AVG processes this query as follows:

- 1. AVG evaluates row 1 of the first partition (Jamaica Plain), but finds no sales within \$50k of this row's sell_price, (\$353k). AVG creates a window that includes this row only, and returns an average of 353k for row 1:

property_key	neighborhood	sell_price	comp_sales
10918	Jamaica Plain	353000	353000
10927	Jamaica Plain	450000	458000
10921	Jamaica Plain	450000	458000
10922	Jamaica Plain	474000	472250

- 2. AVG evaluates row 2 and finds three sell_price values within \$50k of the current row. AVG creates a window that includes these three rows, and returns an average of 458k for

row 2:

property_key	neighborhood	sell_price	comp_sales
10918	Jamaica Plain	353000	353000
10927	Jamaica Plain	450000	458000
10921	Jamaica Plain	450000	458000
10922	Jamaica Plain	474000	472250

- 3. AVG evaluates row 3 and finds the same three sell_price values within \$50k of the current row. AVG creates a window identical to the one before, and returns the same average of 458k for row 3:

property_key	neighborhood	sell_price	comp_sales
10918	Jamaica Plain	353000	353000
10927	Jamaica Plain	450000	458000
10921	Jamaica Plain	450000	458000
10922	Jamaica Plain	474000	472250

- 4. AVG evaluates row 4 and finds four sell_price values within \$50k of the current row. AVG expands its window to include rows 2 through 5, and returns an average of \$472.25k for row 4:

property_key	neighborhood	sell_price	comp_sales
10918	Jamaica Plain	353000	353000
10927	Jamaica Plain	450000	458000
10921	Jamaica Plain	450000	458000
10922	Jamaica Plain	474000	472250
10919	Jamaica Plain	515000	494500
10917	Jamaica Plain	675000	691250

5. In similar fashion, AVG evaluates the remaining rows in this partition. When the function evaluates the first row of the second partition (Roslindale), it resets the window as follows:

property_key	neighborhood	sell_price	comp_sales
10918	Jamaica Plain	353000	353000
10927	Jamaica Plain	450000	458000
10921	Jamaica Plain	450000	458000
10922	Jamaica Plain	474000	472250
10919	Jamaica Plain	515000	494500
10917	Jamaica Plain	675000	691250
10924	Jamaica Plain	675000	691250
10920	Jamaica Plain	705000	691250
10923	Jamaica Plain	710000	691250
10926	Jamaica Plain	875000	887500
10925	Jamaica Plain	900000	887500
10930	Roslindale	300000	300000
10928	Roslindale	422000	436000

AVG () →

Restrictions

If RANGE specifies a constant value, that value's data type and the window's ORDER BY data type must be the same. The following exceptions apply:

- RANGE can specify INTERVAL Year to Month if the window order clause data type is one of following: TIMESTAMP, TIMESTAMP WITH TIMEZONE, or DATE. TIME and TIME WITH TIMEZONE are not supported.
- RANGE can specify INTERVAL Day to Second if the window order clause data is one of following: TIMESTAMP, TIMESTAMP WITH TIMEZONE, DATE, TIME, or TIME WITH TIMEZONE.

The window order clause must specify one of the following data types: NUMERIC, DATE/TIME, FLOAT or INTEGER. This requirement is ignored if the window specifies one of following frames:

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
- RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
- RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

Reporting Aggregates

Some of the analytic functions that take the window-frame-clause are the reporting aggregates. These functions let you compare a partition's aggregate values with detail rows, taking the place of correlated subqueries or joins.

- [AVG\(\)](#)
- [COUNT\(\)](#)
- [MAX\(\)](#) and [MIN\(\)](#)
- [SUM\(\)](#)
- [STDDEV\(\)](#), [STDDEV_POP\(\)](#), and [STDDEV_SAMP\(\)](#)
- [VARIANCE\(\)](#), [VAR_POP\(\)](#), and [VAR_SAMP\(\)](#)

If you use a window aggregate with an empty `OVER()` clause, the analytic function is used as a reporting function, where the entire input is treated as a single partition.

About Standard Deviation and Variance Functions

With standard deviation functions, a low standard deviation indicates that the data points tend to be very close to the mean, whereas high standard deviation indicates that the data points are spread out over a large range of values.

Standard deviation is often graphed and a distributed standard deviation creates the classic bell curve.

Variance functions measure how far a set of numbers is spread out.

Examples

Think of the window for reporting aggregates as a window defined as `UNBOUNDED PRECEDING` and `UNBOUNDED FOLLOWING`. The omission of a window-order-clause makes all rows in the partition also the window (reporting aggregates).

```
=> SELECT deptno, sal, empno, COUNT(sal)
   OVER (PARTITION BY deptno) AS COUNT FROM emp;
deptno | sal | empno | count
-----+-----+-----+-----
```

```
10 | 101 | 1 | 2
10 | 104 | 4 | 2
-----
20 | 110 | 10 | 6
20 | 110 | 9 | 6
20 | 109 | 7 | 6
20 | 109 | 6 | 6
20 | 109 | 8 | 6
20 | 100 | 11 | 6
-----
30 | 105 | 5 | 3
30 | 103 | 3 | 3
30 | 102 | 2 | 3
(11 rows)
```

If the `OVER()` clause in the above query contained a `window-order-clause` (for example, `ORDER BY sal`), it would become a moving window (window aggregate) query with a default window of `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`:

```
=> SELECT deptno, sal, empno, COUNT(sal) OVER (PARTITION BY deptno ORDER BY sal) AS COUNT FROM emp;
deptno | sal | empno | count
-----+-----+-----+-----
10 | 101 | 1 | 1
10 | 104 | 4 | 2
-----
20 | 100 | 11 | 1
20 | 109 | 7 | 4
20 | 109 | 6 | 4
20 | 109 | 8 | 4
20 | 110 | 10 | 6
20 | 110 | 9 | 6
-----
30 | 102 | 2 | 1
30 | 103 | 3 | 2
30 | 105 | 5 | 3
(11 rows)
```

What About `LAST_VALUE`?

You might wonder why you couldn't just use the `LAST_VALUE()` analytic function.

For example, for each employee, get the highest salary in the department:

```
=> SELECT deptno, sal, empno, LAST_VALUE(empno) OVER (PARTITION BY deptno ORDER BY sal) AS lv FROM emp;
```

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	7
20	109	6	7
20	109	8	7
20	110	10	10
20	110	9	10
30	102	2	2
30	103	3	3
30	105	5	5

Due to default window semantics, `LAST_VALUE` does not always return the last value of a partition. If you omit the window-frame-clause from the analytic clause, `LAST_VALUE` operates on this default window. Results, therefore, can seem non-intuitive because the function does not return the bottom of the current partition. It returns the bottom of the window, which continues to change along with the current input row being processed.

Remember the default window:

```
OVER (PARTITION BY deptno ORDER BY sal)
```

is the same as:

```
OVER(PARTITION BY deptno ORDER BY sal ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
```

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	
20	109	6	
20	109	8	
20	110	10	
20	110	9	
30	102	2	2
30	103	3	3
30	105	5	5

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	7
20	109	6	
20	109	8	
20	110	10	
20	110	9	
30	102	2	2
30	103	3	3
30	105	5	5

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	7
20	109	6	7
20	109	8	7
20	110	10	10
20	110	9	10
30	102	2	2
30	103	3	3
30	105	5	5

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	7
20	109	6	7
20	109	8	7
20	110	10	10
20	110	9	10
30	102	2	2
30	103	3	3
30	105	5	5

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	7
20	109	6	7
20	109	8	7
20	110	10	10
20	110	9	10
30	102	2	2
30	103	3	3
30	105	5	5

deptno	sal	empno	lv
10	101	1	1
10	104	4	4
20	100	11	11
20	109	7	7
20	109	6	7
20	109	8	7
20	110	10	10
20	110	9	10
30	102	2	2
30	103	3	3
30	105	5	5

If you want to return the last value of a partition, use UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

```
=> SELECT deptno, sal, empno, LAST_VALUE(empno)
OVER (PARTITION BY deptno ORDER BY sal
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM emp;
```

deptno	sal	empno	lv
10	101	1	4
10	104	4	4
20	100	11	9
20	109	7	9
20	109	6	9
20	109	8	9
20	110	10	9
20	110	9	9
30	102	2	5
30	103	3	5
30	105	5	5

Vertica recommends that you use `LAST_VALUE` with the `window-order-clause` to produce deterministic results.

In the following example, `empno` 6, 7, and 8 have the same salary, so they are in adjacent rows. `empno` 8 appears first in this case but the order is not guaranteed.

deptno	sal	empno	lv
10	101	1	4
10	104	4	4
20	100	11	7
20	109	8	7
20	109	6	7
20	109	7	7
30	102	2	5
30	103	3	5
30	105	5	5

Notice in the output above, the last value is 7, which is the last row from the partition `deptno = 20`. If the rows have a different order, then the function returns a different value:

deptno	sal	empno	lv
10	101	1	4
10	104	4	4
20	100	11	6
20	109	8	6
20	109	7	6
20	109	6	6
30	102	2	5
30	103	3	5
30	105	5	5

Now the last value is 6, which is the last row from the partition deptno = 20. The solution is to add a unique key to the sort order. Even if the order of the query changes, the result will always be the same, and so deterministic.

```
=> SELECT deptno, sal, empno, LAST_VALUE(empno)
OVER (PARTITION BY deptno ORDER BY sal, empno
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) as lv
FROM emp;
```

deptno	sal	empno	lv
10	101	1	4
10	104	4	4
20	100	11	8
20	109	6	8
20	109	7	8
20	109	8	8
30	102	2	5
30	103	3	5
30	105	5	5

Notice how the rows are now ordered by empno, the last value stays at 8, and it does not matter the order of the query.

Named Windows

An analytic function's OVER clause can reference a named window, which encapsulates one or more window clauses: a window partition (PARTITION BY) clause and (optionally) a window order (ORDER BY) clause. Named windows can be useful when you write queries that invoke

multiple analytic functions with similar OVER clause syntax—for example, they use the same partition clauses.

A query names a window as follows:

```
WINDOW window-name AS ( window-partition-clause [window-order-clause] );
```

The same query can name and reference multiple windows. All window names must be unique within the same query.

Examples

The following query invokes two analytic functions, [RANK](#) and [DENSE_RANK](#). Because the two functions use the same partition and order clauses, the query names a window w that specifies both clauses. The two functions reference this window as follows:

```
=> SELECT employee_region region, employee_key, annual_salary,
        RANK() OVER w Rank,
        DENSE_RANK() OVER w "Dense Rank"
        FROM employee_dimension WINDOW w AS (PARTITION BY employee_region ORDER BY annual_salary);
```

region	employee_key	annual_salary	Rank	Dense Rank
West	5248	1200	1	1
West	6880	1204	2	2
West	5700	1214	3	3
West	9857	1218	4	4
West	6014	1218	4	4
West	9221	1220	6	5
West	7646	1222	7	6
West	6621	1222	7	6
West	6488	1224	9	7
West	7659	1226	10	8
West	7432	1226	10	8
West	9905	1226	10	8
West	9021	1228	13	9
West	7855	1228	13	9
West	7119	1230	15	10
...				

If the named window omits an order clause, the query's OVER clauses can specify their own order clauses. For example, you can modify the previous query so each function uses a different order clause. The named window is defined so it includes only a partition clause:

```
=> SELECT employee_region region, employee_key, annual_salary,
        RANK() OVER (w ORDER BY annual_salary DESC) Rank,
        DENSE_RANK() OVER (w ORDER BY annual_salary ASC) "Dense Rank"
        FROM employee_dimension WINDOW w AS (PARTITION BY employee_region);
```

region	employee_key	annual_salary	Rank	Dense Rank
West	5248	1200	2795	1
West	6880	1204	2794	2
West	5700	1214	2793	3
West	6014	1218	2791	4

West	9857	1218	2791	4
West	9221	1220	2790	5
West	6621	1222	2788	6
West	7646	1222	2788	6
West	6488	1224	2787	7
West	7432	1226	2784	8
West	9905	1226	2784	8
West	7659	1226	2784	8
West	7855	1228	2782	9
West	9021	1228	2782	9
West	7119	1230	2781	10
...				

Similarly, an `OVER` clause specifies a named window can also specify a [window frame clause](#), provided the named window includes an order clause. This can be useful inasmuch as you cannot define a named windows to include a window frame clause.

For example, the following query defines a window that encapsulates partitioning and order clauses. The `OVER` clause invokes this window and also includes a window frame clause:

```
=> SELECT deptno, sal, empno, COUNT(*) OVER (w ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS count
FROM emp WINDOW w AS (PARTITION BY deptno ORDER BY sal);
deptno | sal | empno | count
-----+-----+-----+-----
 10 | 101 | 1 | 1
 10 | 104 | 4 | 2
 20 | 100 | 11 | 1
 20 | 109 | 8 | 2
 20 | 109 | 6 | 3
 20 | 109 | 7 | 3
 20 | 110 | 10 | 3
 20 | 110 | 9 | 3
 30 | 102 | 2 | 1
 30 | 103 | 3 | 2
 30 | 105 | 5 | 3
(11 rows)
```

Recursive Window References

A `WINDOW` clause can reference another window that is already named. For example, because named window `w1` is defined before `w2`, the `WINDOW` clause that defines `w2` can reference `w1`:

```
=> SELECT RANK() OVER(w1 ORDER BY sal DESC), RANK() OVER w2
FROM EMP WINDOW w1 AS (PARTITION BY deptno), w2 AS (w1 ORDER BY sal);
```

Restrictions

- An `OVER` clause can reference only one named window.
- Each `WINDOW` clause within the same query must have a unique name.

Analytic Query Examples

- [Calculating a Median Value](#)
- [Getting Price Differential for Two Stocks](#)
- [Calculating the Moving Average](#)
- [Getting Latest Bid and Ask Results](#)

Calculating a Median Value

A median is a numerical value that separates the higher half of a sample from the lower half. For example, you can retrieve the median of a finite list of numbers by arranging all observations from lowest value to highest value and then picking the middle one.

If the number of observations is even, then there is no single middle value; the median is the mean (average) of the two middle values.

The following example uses this table:

```
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT);
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

You can use the analytic function **MEDIAN** to calculate the median of all sales in this table. In the following query, the function's **OVER** clause is empty, so the query returns the same aggregated value for each row of the result set:

```
=> SELECT name, sales, MEDIAN(sales) OVER() AS median FROM allsales;
 name | sales | median
-----+-----+-----
 G    |    10 |    20
 C    |    15 |    20
 D    |    20 |    20
 B    |    20 |    20
 F    |    40 |    20
 E    |    50 |    20
 A    |    60 |    20
(7 rows)
```

You can modify this query to group sales by state and obtain the median for each one. To do so, include a window partition clause in the OVER clause:

```
=> SELECT state, name, sales, MEDIAN(sales) OVER(partition by state) AS median FROM allsales;
```

state	name	sales	median
MA	G	10	35
MA	D	20	35
MA	E	50	35
MA	A	60	35
NY	C	15	20
NY	B	20	20
NY	F	40	20

(7 rows)

Getting Price Differential for Two Stocks

The following subquery selects out two stocks of interest. The outer query uses the LAST_VALUE() and OVER() components of analytics, with IGNORE NULLS.

Schema

```
DROP TABLE Ticks CASCADE;
```

```
CREATE TABLE Ticks (ts TIMESTAMP, Stock varchar(10), Bid float);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:23:54', 'abc', 10.12);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:23:58', 'abc', 10.34);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:23:59', 'abc', 10.75);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:25:15', 'abc', 11.98);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:25:16', 'abc');
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:25:22', 'xyz', 45.16);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:25:27', 'xyz', 49.33);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:31:12', 'xyz', 65.25);
```

```
INSERT INTO Ticks VALUES('2011-07-12 10:31:15', 'xyz');
```

```
COMMIT;
```

ticks Table

```
=> SELECT * FROM ticks;
```

ts	stock	bid
2011-07-12 10:23:59	abc	10.75
2011-07-12 10:25:22	xyz	45.16
2011-07-12 10:23:58	abc	10.34
2011-07-12 10:25:27	xyz	49.33
2011-07-12 10:23:54	abc	10.12

```
2011-07-12 10:31:15 | xyz |  
2011-07-12 10:25:15 | abc | 11.98  
2011-07-12 10:25:16 | abc |  
2011-07-12 10:31:12 | xyz | 65.25  
(9 rows)
```

Query

```
=> SELECT ts, stock, bid, last_value(price1 IGNORE NULLS)  
      OVER(ORDER BY ts) - last_value(price2 IGNORE NULLS)  
      OVER(ORDER BY ts) as price_diff  
FROM  
(SELECT ts, stock, bid,  
  CASE WHEN stock = 'abc' THEN bid ELSE NULL END AS price1,  
  CASE WHEN stock = 'xyz' THEN bid ELSE NULL END AS price2  
  FROM ticks  
  WHERE stock IN ('abc','xyz')  
 ) v1  
ORDER BY ts;  
      ts          | stock | bid  | price_diff  
-----+-----+-----+-----  
2011-07-12 10:23:54 | abc  | 10.12 |  
2011-07-12 10:23:58 | abc  | 10.34 |  
2011-07-12 10:23:59 | abc  | 10.75 |  
2011-07-12 10:25:15 | abc  | 11.98 |  
2011-07-12 10:25:16 | abc  |      |  
2011-07-12 10:25:22 | xyz  | 45.16 | -33.18  
2011-07-12 10:25:27 | xyz  | 49.33 | -37.35  
2011-07-12 10:31:12 | xyz  | 65.25 | -53.27  
2011-07-12 10:31:15 | xyz  |      | -53.27  
(9 rows)
```

Calculating the Moving Average

Calculate a 40-second moving average of bids for one stock. This examples uses the [ticks](#) table schema.

Query

```
=> SELECT ts, bid, AVG(bid)  
      OVER(ORDER BY ts  
          RANGE BETWEEN INTERVAL '40 seconds'  
          PRECEDING AND CURRENT ROW)  
FROM ticks  
WHERE stock = 'abc'  
GROUP BY bid, ts  
ORDER BY ts;
```

```
      ts          | bid |      ?column?
-----+-----+-----
2011-07-12 10:23:54 | 10.12 |          10.12
2011-07-12 10:23:58 | 10.34 |          10.23
2011-07-12 10:23:59 | 10.75 | 10.40333333333333
2011-07-12 10:25:15 | 11.98 |          11.98
2011-07-12 10:25:16 |      |          11.98
(5 rows)
```

```
DROP TABLE Ticks CASCADE;
```

```
CREATE TABLE Ticks (ts TIMESTAMP, Stock varchar(10), Bid float);
INSERT INTO Ticks VALUES('2011-07-12 10:23:54', 'abc', 10.12);
INSERT INTO Ticks VALUES('2011-07-12 10:23:58', 'abc', 10.34);
INSERT INTO Ticks VALUES('2011-07-12 10:23:59', 'abc', 10.75);
INSERT INTO Ticks VALUES('2011-07-12 10:25:15', 'abc', 11.98);
INSERT INTO Ticks VALUES('2011-07-12 10:25:16', 'abc');
INSERT INTO Ticks VALUES('2011-07-12 10:25:22', 'xyz', 45.16);
INSERT INTO Ticks VALUES('2011-07-12 10:25:27', 'xyz', 49.33);
INSERT INTO Ticks VALUES('2011-07-12 10:31:12', 'xyz', 65.25);
INSERT INTO Ticks VALUES('2011-07-12 10:31:15', 'xyz');

COMMIT;
```

Getting Latest Bid and Ask Results

The following query fills in missing NULL values to create a full book order showing latest bid and ask price and size, by vendor id. Original rows have values for (typically) one price and one size, so use `last_value` with "ignore nulls" to find the most recent non-null value for the other pair each time there is an entry for the ID. `Sequenceno` provides a unique total ordering.

Schema:

```
=> CREATE TABLE bookorders(
  vendorid VARCHAR(100),
  date TIMESTAMP,
  sequenceno INT,
  askprice FLOAT,
  asksize INT,
  bidprice FLOAT,
  bidsize INT);
=> INSERT INTO bookorders VALUES('3325XPK','2011-07-12 10:23:54', 1, 10.12, 55, 10.23, 59);
=> INSERT INTO bookorders VALUES('3345XPZ','2011-07-12 10:23:55', 2, 10.55, 58, 10.75, 57);
=> INSERT INTO bookorders VALUES('445XPKF','2011-07-12 10:23:56', 3, 10.22, 43, 54);
=> INSERT INTO bookorders VALUES('445XPKF','2011-07-12 10:23:57', 3, 10.22, 59, 10.25, 61);
=> INSERT INTO bookorders VALUES('3425XPY','2011-07-12 10:23:58', 4, 11.87, 66, 11.90, 66);
=> INSERT INTO bookorders VALUES('3727XVK','2011-07-12 10:23:59', 5, 11.66, 51, 11.67, 62);
=> INSERT INTO bookorders VALUES('5325XYZ','2011-07-12 10:24:01', 6, 15.05, 44, 15.10, 59);
=> INSERT INTO bookorders VALUES('3675XVS','2011-07-12 10:24:05', 7, 15.43, 47, 58);
```

```
=> INSERT INTO bookorders VALUES('8972VUG','2011-07-12 10:25:15', 8, 14.95, 52, 15.11, 57);  
COMMIT;
```

Query:

```
=> SELECT  
  sequenceno Seq,  
  date "Time",  
  vendorid ID,  
  LAST_VALUE (bidprice IGNORE NULLS)  
    OVER (PARTITION BY vendorid ORDER BY sequenceno  
          ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)  
  AS "Bid Price",  
  LAST_VALUE (bidsize IGNORE NULLS)  
    OVER (PARTITION BY vendorid ORDER BY sequenceno  
          ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)  
  AS "Bid Size",  
  LAST_VALUE (askprice IGNORE NULLS)  
    OVER (PARTITION BY vendorid ORDER BY sequenceno  
          ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)  
  AS "Ask Price",  
  LAST_VALUE (asksize IGNORE NULLS)  
    OVER (PARTITION BY vendorid order by sequenceno  
          ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )  
  AS "Ask Size"  
FROM bookorders  
ORDER BY sequenceno;
```

Seq	Time	ID	Bid Price	Bid Size	Ask Price	Ask Size
1	2011-07-12 10:23:54	3325XPK	10.23	59	10.12	55
2	2011-07-12 10:23:55	3345XPZ	10.75	57	10.55	58
3	2011-07-12 10:23:57	445XPKF	10.25	61	10.22	59
3	2011-07-12 10:23:56	445XPKF	54		10.22	43
4	2011-07-12 10:23:58	3425XPY	11.9	66	11.87	66
5	2011-07-12 10:23:59	3727XVK	11.67	62	11.66	51
6	2011-07-12 10:24:01	5325XYZ	15.1	59	15.05	44
7	2011-07-12 10:24:05	3675XVS	58		15.43	47
8	2011-07-12 10:25:15	8972VUG	15.11	57	14.95	52

(9 rows)

Event-Based Windows

Event-based windows let you break time series data into windows that border on significant events within the data. This is especially relevant in financial data where analysis often focuses on specific events as triggers to other activity.

Vertica provides two event-based window functions that are not part of the SQL-99 standard:

- [CONDITIONAL_CHANGE_EVENT](#) assigns an event window number to each row, starting from 0, and increments by 1 when the result of evaluating the argument expression on the current row differs from that on the previous row. This function is similar to the analytic function [ROW_NUMBER](#), which assigns a unique number, sequentially, starting from 1, to each row within a partition.
- [CONDITIONAL_TRUE_EVENT](#) assigns an event window number to each row, starting from 0, and increments the number by 1 when the result of the boolean argument expression evaluates true.

Both functions are described in greater detail below.

Note: [CONDITIONAL_CHANGE_EVENT](#) and [CONDITIONAL_TRUE_EVENT](#) do not support [window framing](#).

Example schema

The examples on this page use the following schema:

```
CREATE TABLE TickStore3 (ts TIMESTAMP, symbol VARCHAR(8), bid FLOAT);
CREATE PROJECTION TickStore3_p (ts, symbol, bid) AS SELECT * FROM TickStore3 ORDER BY ts, symbol, bid
UNSEGMENTED ALL NODES;
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:00', 'XYZ', 10.0);
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:03', 'XYZ', 11.0);
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:06', 'XYZ', 10.5);
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:09', 'XYZ', 11.0);
COMMIT;
```

Using [CONDITIONAL_CHANGE_EVENT](#)

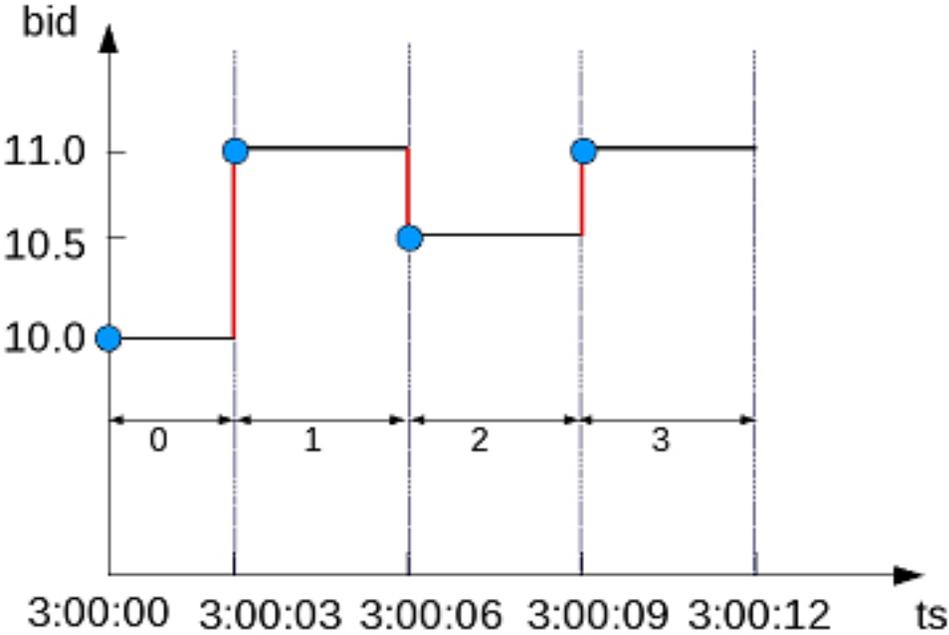
The analytical function [CONDITIONAL_CHANGE_EVENT](#) returns a sequence of integers indicating event window numbers, starting from 0. The function increments the event window number when the result of evaluating the function expression on the current row differs from the previous value.

In the following example, the first query returns all records from the TickStore3 table. The second query uses the [CONDITIONAL_CHANGE_EVENT](#) function on the bid column. Since each bid row value is different from the previous value, the function increments the window ID from 0 to 3:

<pre>SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;</pre>	==>	<pre>SELECT CONDITIONAL_CHANGE_EVENT(bid) OVER(ORDER BY ts) FROM Tickstore3;</pre>														
<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; border-bottom: 1px dashed black;">ts</th> <th style="text-align: left; border-bottom: 1px dashed black;">symbol</th> <th style="text-align: left; border-bottom: 1px dashed black;">bid</th> </tr> </thead> <tbody> <tr> <td style="border-bottom: 1px dashed black;">2009-01-01 03:00:00</td> <td style="border-bottom: 1px dashed black;">XYZ</td> <td style="border-bottom: 1px dashed black;">10</td> </tr> </tbody> </table>	ts	symbol	bid	2009-01-01 03:00:00	XYZ	10		<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; border-bottom: 1px dashed black;">ts</th> <th style="text-align: left; border-bottom: 1px dashed black;">symbol</th> <th style="text-align: left; border-bottom: 1px dashed black;">bid</th> <th style="text-align: left; border-bottom: 1px dashed black;">cce</th> </tr> </thead> <tbody> <tr> <td style="border-bottom: 1px dashed black;">2009-01-01 03:00:00</td> <td style="border-bottom: 1px dashed black;">XYZ</td> <td style="border-bottom: 1px dashed black;">10</td> <td style="border-bottom: 1px dashed black;">0</td> </tr> </tbody> </table>	ts	symbol	bid	cce	2009-01-01 03:00:00	XYZ	10	0
ts	symbol	bid														
2009-01-01 03:00:00	XYZ	10														
ts	symbol	bid	cce													
2009-01-01 03:00:00	XYZ	10	0													

2009-01-01 03:00:03 XYZ 11	2009-01-01 03:00:03 XYZ 11 1
2009-01-01 03:00:06 XYZ 10.5	2009-01-01 03:00:06 XYZ 10.5 2
2009-01-01 03:00:09 XYZ 11	2009-01-01 03:00:09 XYZ 11 3
(4 rows)	(4 rows)

The following figure is a graphical illustration of the change in the bid price. Each value is different from its previous one, so the window ID increments for each time slice:

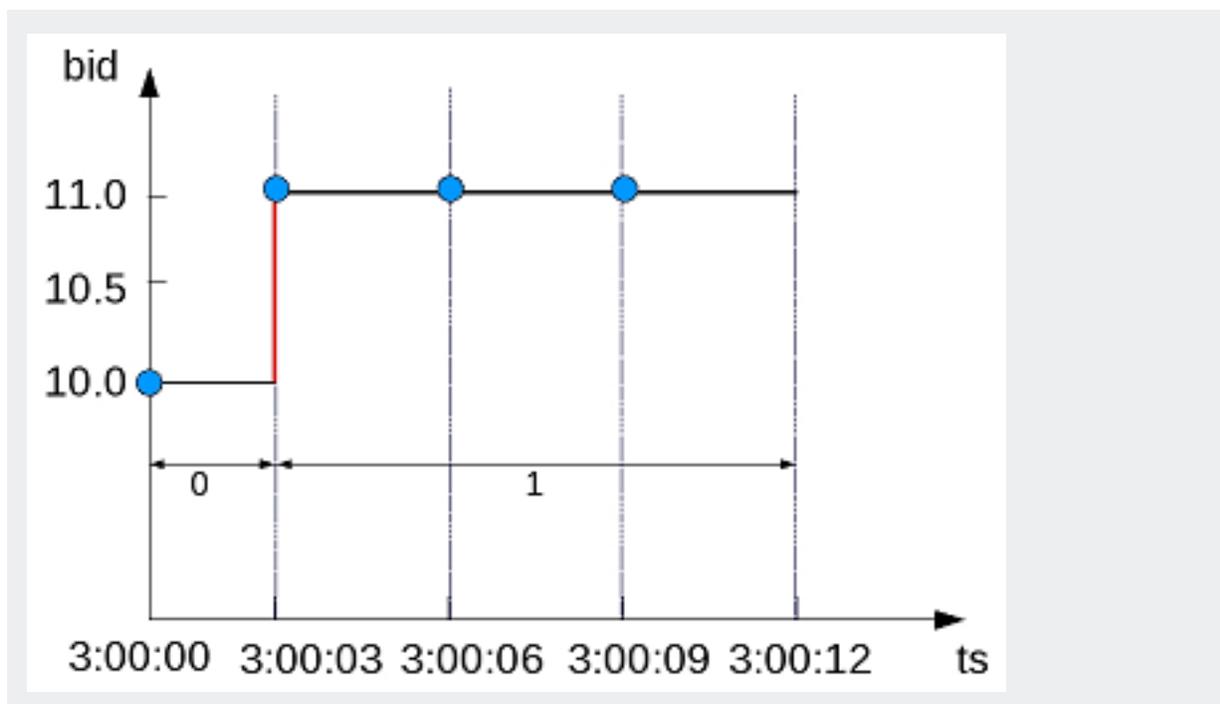


So the window ID starts at 0 and increments at every change in from the previous value.

In this example, the bid price changes from \$10 to \$11 in the second row, but then stays the same. `CONDITIONAL_CHANGE_EVENT` increments the event window ID in row 2, but not subsequently:

<code>SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;</code>	<code>SELECT CONDITIONAL_CHANGE_EVENT(bid) OVER(ORDER BY ts) FROM Tickstore3;</code>																																			
<table border="1"> <thead> <tr> <th>ts</th> <th>symbol</th> <th>bid</th> </tr> </thead> <tbody> <tr><td>2009-01-01 03:00:00</td><td>XYZ</td><td>10</td></tr> <tr><td>2009-01-01 03:00:03</td><td>XYZ</td><td>11</td></tr> <tr><td>2009-01-01 03:00:06</td><td>XYZ</td><td>11</td></tr> <tr><td>2009-01-01 03:00:09</td><td>XYZ</td><td>11</td></tr> </tbody> </table>	ts	symbol	bid	2009-01-01 03:00:00	XYZ	10	2009-01-01 03:00:03	XYZ	11	2009-01-01 03:00:06	XYZ	11	2009-01-01 03:00:09	XYZ	11	<p>==></p> <table border="1"> <thead> <tr> <th>ts</th> <th>symbol</th> <th>bid</th> <th>cce</th> </tr> </thead> <tbody> <tr><td>2009-01-01 03:00:00</td><td>XYZ</td><td>10</td><td>0</td></tr> <tr><td>2009-01-01 03:00:03</td><td>XYZ</td><td>11</td><td>1</td></tr> <tr><td>2009-01-01 03:00:06</td><td>XYZ</td><td>11</td><td>1</td></tr> <tr><td>2009-01-01 03:00:09</td><td>XYZ</td><td>11</td><td>1</td></tr> </tbody> </table>	ts	symbol	bid	cce	2009-01-01 03:00:00	XYZ	10	0	2009-01-01 03:00:03	XYZ	11	1	2009-01-01 03:00:06	XYZ	11	1	2009-01-01 03:00:09	XYZ	11	1
ts	symbol	bid																																		
2009-01-01 03:00:00	XYZ	10																																		
2009-01-01 03:00:03	XYZ	11																																		
2009-01-01 03:00:06	XYZ	11																																		
2009-01-01 03:00:09	XYZ	11																																		
ts	symbol	bid	cce																																	
2009-01-01 03:00:00	XYZ	10	0																																	
2009-01-01 03:00:03	XYZ	11	1																																	
2009-01-01 03:00:06	XYZ	11	1																																	
2009-01-01 03:00:09	XYZ	11	1																																	

The following figure is a graphical illustration of the change in the bid price at 3:00:03 only. The price stays the same at 3:00:06 and 3:00:09, so the window ID remains at 1 for each time slice after the change:



Using CONDITIONAL_TRUE_EVENT

Like `CONDITIONAL_CHANGE_EVENT`, the analytic function `CONDITIONAL_TRUE_EVENT` also returns a sequence of integers indicating event window numbers, starting from 0. The two functions differ as follows:

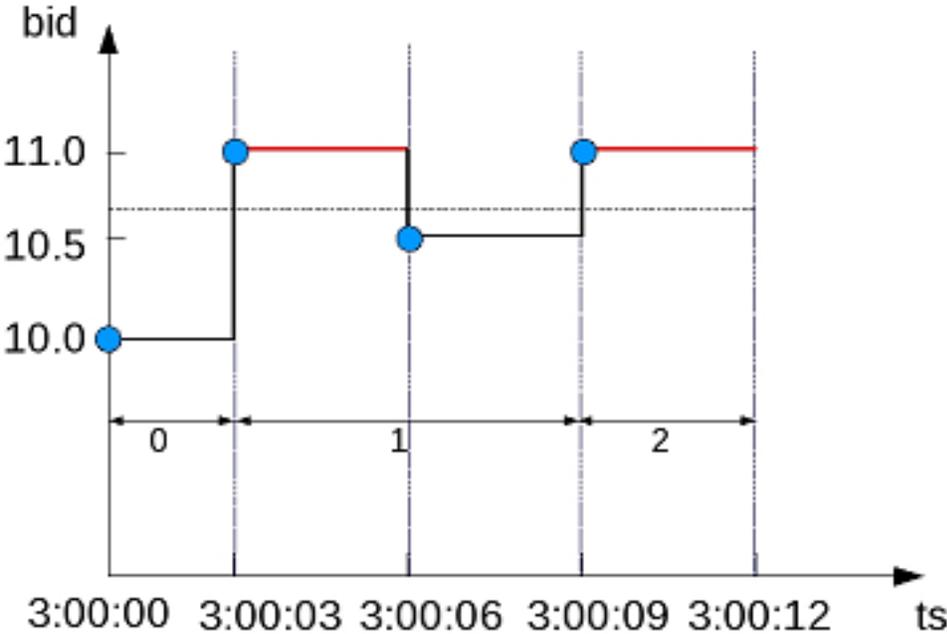
- `CONDITIONAL_TRUE_EVENT` increments the window ID each time its expression evaluates to true.
- `CONDITIONAL_CHANGE_EVENT` increments on a comparison expression with the previous value.

In the following example, the first query returns all records from the `TickStore3` table. The second query uses `CONDITIONAL_TRUE_EVENT` to test whether the current bid is greater than a given value (10.6). Each time the expression tests true, the function increments the window ID. The first time the function increments the window ID is on row 2, when the value is 11. The expression tests false for the next row (value is not greater than 10.6), so the function does not increment the event window ID. In the final row, the expression is true for the given condition, and the function increments the window:

<pre>SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;</pre>		<pre>SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6) OVER(ORDER BY ts) FROM Tickstore3;</pre>
ts symbol bid	==>	ts symbol bid cte-----

<pre>-- 2009-01-01 03:00:00 XYZ 10 2009-01-01 03:00:03 XYZ 11 2009-01-01 03:00:06 XYZ 10.5 2009-01-01 03:00:09 XYZ 11</pre>	<pre>2009-01-01 03:00:00 XYZ 10 0 2009-01-01 03:00:03 XYZ 11 1 2009-01-01 03:00:06 XYZ 10.5 1 2009-01-01 03:00:09 XYZ 11 2</pre>
---	--

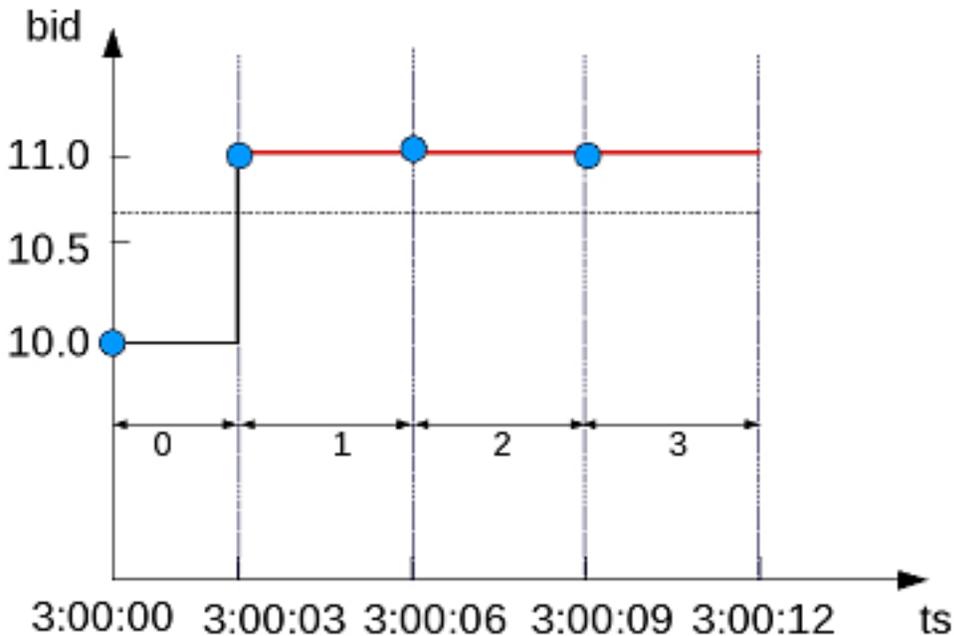
The following figure is a graphical illustration that shows the bid values and window ID changes. Because the bid value is greater than \$10.6 on only the second and fourth time slices (3:00:03 and 3:00:09), the window ID returns <0,1,1,2>:



In the following example, the first query returns all records from the TickStore3 table, ordered by the tickstore values (ts). The second query uses `CONDITIONAL_TRUE_EVENT` to increment the window ID each time the bid value is greater than 10.6. The first time the function increments the event window ID is on row 2, where the value is 11. The window ID then increments each time after that, because the expression `(bid > 10.6)` tests true for each time slice:

<pre>SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;</pre>	<pre>SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6) OVER(ORDER BY ts) FROM Tickstore3;</pre>
<pre>ts symbol bid -----+-----+----- 2009-01-01 03:00:00 XYZ 10 2009-01-01 03:00:03 XYZ 11 2009-01-01 03:00:06 XYZ 11 2009-01-01 03:00:09 XYZ 11</pre>	<pre>ts symbol bid cte -----+-----+-----+----- 2009-01-01 03:00:00 XYZ 10 0 2009-01-01 03:00:03 XYZ 11 1 2009-01-01 03:00:06 XYZ 11 2 2009-01-01 03:00:09 XYZ 11 3</pre>

The following figure is a graphical illustration that shows the bid values and window ID changes. The bid value is greater than 10.6 on the second time slice (3:00:03) and remains for the remaining two time slices. The function increments the event window ID each time because the expression tests true:



Advanced Use of Event-Based Windows

In event-based window functions, the condition expression accesses values from the current row only. To access a previous value, you can use a more powerful event-based window that allows the window event condition to include previous data points. For example, analytic function `LAG(x, n)` retrieves the value of column `x` in the `n`th to last input record. In this case, `LAG` shares the `OVER` specifications of the `CONDITIONAL_CHANGE_EVENT` or `CONDITIONAL_TRUE_EVENT` function expression.

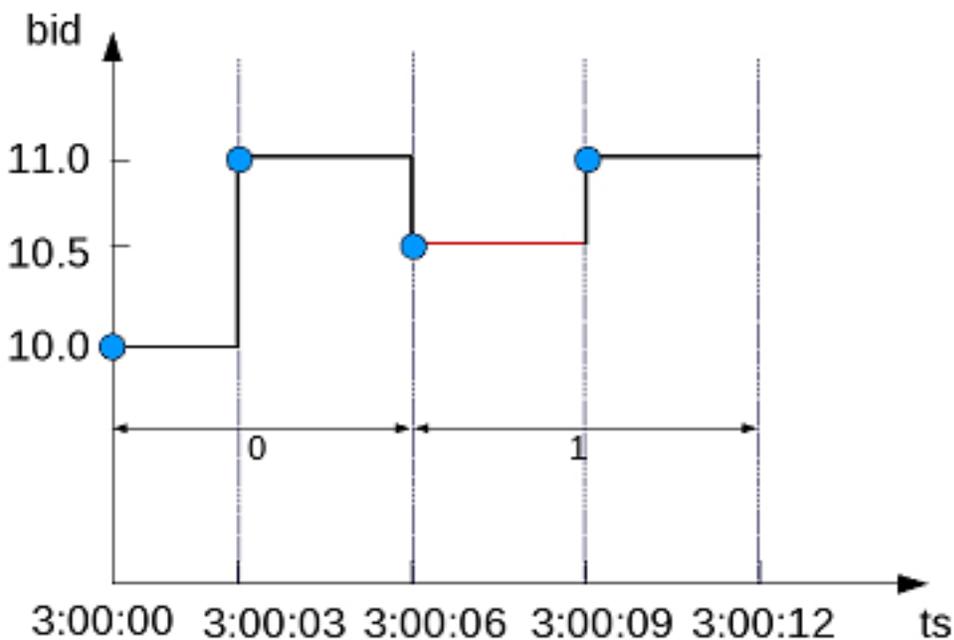
In the following example, the first query returns all records from the `TickStore3` table. The second query uses `CONDITIONAL_TRUE_EVENT` with the `LAG` function in its boolean expression. In this case, `CONDITIONAL_TRUE_EVENT` increments the event window ID each time the bid value on the current row is less than the previous value. The first time `CONDITIONAL_TRUE_EVENT` increments the window ID starts on the third time slice, when the expression tests true. The current value (10.5) is less than the previous value. The window ID is not incremented in the last row because the final value is greater than the previous row:

```
SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;
```

```
SELECT CONDITIONAL_TRUE_EVENT(bid < LAG(bid))  
OVER(ORDER BY ts) FROM Tickstore3;
```

ts	symbol	bid	ts	symbol	bid	cte
2009-01-01 03:00:00	XYZ	10	2009-01-01 03:00:00	XYZ	10	0
2009-01-01 03:00:03	XYZ	11	2009-01-01 03:00:03	XYZ	11	0
2009-01-01 03:00:06	XYZ	10.5	2009-01-01 03:00:06	XYZ	10.5	1
2009-01-01 03:00:09	XYZ	11	2009-01-01 03:00:09	XYZ	11	1

The following figure illustrates the second query above. When the bid price is less than the previous value, the window ID gets incremented, which occurs only in the third time slice (3:00:06):



See Also

- [Sessionization with Event-Based Windows](#)
- [Time Series Analytics](#)

Sessionization with Event-Based Windows

Sessionization, a special case of event-based windows, is a feature often used to analyze click streams, such as identifying web browsing sessions from recorded web clicks.

In Vertica, given an input clickstream table, where each row records a Web page click made by a particular user (or IP address), the sessionization computation attempts to identify Web

browsing sessions from the recorded clicks by grouping the clicks from each user based on the time-intervals between the clicks. If two clicks from the same user are made too far apart in time, as defined by a time-out threshold, the clicks are treated as though they are from two different browsing sessions.

Example Schema

The examples in this topic use the following WebClicks schema to represent a simple clickstream table:

```
CREATE TABLE WebClicks(userId INT, timestamp TIMESTAMP);
INSERT INTO WebClicks VALUES (1, '2009-12-08 3:00:00 pm');
INSERT INTO WebClicks VALUES (1, '2009-12-08 3:00:25 pm');
INSERT INTO WebClicks VALUES (1, '2009-12-08 3:00:45 pm');
INSERT INTO WebClicks VALUES (1, '2009-12-08 3:01:45 pm');
INSERT INTO WebClicks VALUES (2, '2009-12-08 3:02:45 pm');
INSERT INTO WebClicks VALUES (2, '2009-12-08 3:02:55 pm');
INSERT INTO WebClicks VALUES (2, '2009-12-08 3:03:55 pm');
COMMIT;
```

The input table `WebClicks` contains the following rows:

```
=> SELECT * FROM WebClicks;
  userId |      timestamp
-----+-----
      1 | 2009-12-08 15:00:00
      1 | 2009-12-08 15:00:25
      1 | 2009-12-08 15:00:45
      1 | 2009-12-08 15:01:45
      2 | 2009-12-08 15:02:45
      2 | 2009-12-08 15:02:55
      2 | 2009-12-08 15:03:55
(7 rows)
```

In the following query, sessionization performs computation on the SELECT list columns, showing the difference between the current and previous timestamp value using `LAG()`. It evaluates to true and increments the window ID when the difference is greater than 30 seconds.

```
=> SELECT userId, timestamp,
  CONDITIONAL_TRUE_EVENT(timestamp - LAG(timestamp) > '30 seconds')
  OVER(PARTITION BY userId ORDER BY timestamp) AS session FROM WebClicks;
  userId |      timestamp | session
-----+-----+-----
      1 | 2009-12-08 15:00:00 |      0
      1 | 2009-12-08 15:00:25 |      0
      1 | 2009-12-08 15:00:45 |      0
      1 | 2009-12-08 15:01:45 |      1
      2 | 2009-12-08 15:02:45 |      0
      2 | 2009-12-08 15:02:55 |      0
      2 | 2009-12-08 15:03:55 |      1
(7 rows)
```

In the output, the session column contains the window ID from the `CONDITIONAL_TRUE_EVENT` function. The window ID evaluates to true on row 4 (timestamp 15:01:45), and the ID that follows row 4 is zero because it is the start of a new partition (for user ID 2), and that row does not evaluate to true until the last line in the output.

You might want to give users different time-out thresholds. For example, one user might have a slower network connection or be multi-tasking, while another user might have a faster connection and be focused on a single Web site, doing a single task.

To compute an adaptive time-out threshold based on the last 2 clicks, use `CONDITIONAL_TRUE_EVENT` with `LAG` to return the average time between the last 2 clicks with a grace period of 3 seconds:

```
=> SELECT userId, timestamp, CONDITIONAL_TRUE_EVENT(timestamp - LAG(timestamp) >
(LAG(timestamp, 1) - LAG(timestamp, 3)) / 2 + '3 seconds')
OVER(PARTITION BY userId ORDER BY timestamp) AS session
FROM WebClicks;
```

userId	timestamp	session
2	2009-12-08 15:02:45	0
2	2009-12-08 15:02:55	0
2	2009-12-08 15:03:55	0
1	2009-12-08 15:00:00	0
1	2009-12-08 15:00:25	0
1	2009-12-08 15:00:45	0
1	2009-12-08 15:01:45	1

(7 rows)

Note: You cannot define a moving window in time series data. For example, if the query is evaluating the first row and there's no data, it will be the current row. If you have a lag of 2, no results are returned until the third row.

See Also

- [Event-Based Windows](#)
- [CONDITIONAL_TRUE_EVENT \[Analytic\]](#)

Machine Learning for Predictive Analytics

Vertica provides a number of machine learning functions for performing in-database analysis. These functions can perform a number of data preparation and predictive tasks—for example:

- Balance an uneven distribution of classes — See [Balancing Imbalanced Data](#) for an example.
- Remove outliers from your data — See [Detecting Outliers](#) for an example.
- Impute missing values in a data set — See [Imputing Missing Values](#) for an example.
- Normalize data to organize different scales of numeric data to an equivalent scale — See [Normalizing Data](#) for an example.
- Create a sample of a larger data set — See [Sampling Data](#) for an example.
- Use regression algorithms to make predictions about features in your data set and an observed value response — See [Regression Algorithms](#) for more information.
- Use classification algorithms to assign items in a data set to different categories — See [Classification Algorithms](#) for more information.
- Use the clustering algorithm to partition data using k-means clustering — See [Clustering Algorithms](#) for more information.

For more information about specific machine learning functions see [Machine Learning Functions](#).

Downloading the Machine Learning Example Data

You need several data sets to run the machine learning examples. You can download these data sets from the Vertica Github repository. These examples introduce the machine learning functionality provide by Vertica.

You can download the example data in either of two ways:

- Download the ZIP file. Extract the contents of the file into a directory.
- Clone the Vertica Machine Learning Github repository. Using a terminal window, run the following command:

```
$ git clone https://github.com/vertica/Machine-Learning-Examples
```

Loading the Example Data

You can load the example data by either:

- Copying and pasting the DDL and DML operations in `load_ml_data.sql` in a vsql prompt or another Vertica client.
- Running the following command from a terminal window in the Machine-Learning-Examples directory:

```
$ /opt/vertica/bin/vsql -d <name of your database> -f load_ml_data.sql
```

Example Data Descriptions

The repository contains the following data sets.

Name	Description
agar_dish	Synthetic data set meant to represent clustering of bacteria on an agar dish. Contains the following columns: id, x-coordinate, and y-coordinate.
agar_dish_2	125 rows sampled randomly from the original 500 rows of the agar_dish data set.
agar_dish_1	375 rows sampled randomly from the original 500 rows of the agar_dish data set.
baseball	Contains statistics from a fictional baseball league. The statistics included are: first name, last name, date of birth, team name, homeruns, hits, batting average, and salary.
faithful	Wait times between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

Name	Description
	<p>Reference</p> <p>Härdle, W. (1991) <i>Smoothing Techniques with Implementation in S</i>. New York: Springer.</p> <p>Azzalini, A. and Bowman, A. W. (1990). A look at some data on the Old Faithful geyser. <i>Applied Statistics</i> 39, 357–365.</p>
faithful_testing	Roughly 60% of the original 272 rows of the faithful data set.
faithful_training	Roughly 40% of the original 272 rows of the faithful data set.
house84	<p>The house84 data set includes votes for each of the U.S. House of Representatives Congress members on 16 votes. Contains the following columns: id, party, vote1, vote2, vote3, vote4, vote5, vote6, vote7, vote8, vote9, vote10, vote11, vote12, vote13, vote14, vote15, vote16.</p> <p>Reference</p> <p>Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc. Washington, D.C., 1985.</p>
iris	<p>The iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.</p> <p>Reference</p> <p>Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) <i>The New S Language</i>. Wadsworth & Brooks/Cole.</p>
iris1	90 rows sampled randomly from the original 150 rows in the iris data set.
iris2	60 rows sampled randomly from the original 150 rows in the iris data set.
mtcars	<p>The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).</p> <p>Reference</p> <p>Henderson and Velleman (1981), Building multiple regression models interactively. <i>Biometrics</i>, 37, 391–411.</p>

Name	Description
salary_data	Contains fictional employee data. The data included are: employee id, first name, last name, years worked, and current salary.
transaction_data	Contains fictional credit card transactions with a BOOLEAN column indicating whether there was fraud associated with the transaction. The data included are: first name, last name, store, cost, and fraud.

Data Preparation

Before you can analyze your data, you must prepare it. You can do the following data preparation tasks in Vertica:

- [Balancing Imbalanced Data](#)
- [Detecting Outliers](#)
- [Imputing Missing Values](#)
- [Normalizing Data](#)
- [Sampling Data](#)

Balancing Imbalanced Data

Imbalanced data occurs when an uneven distribution of classes occurs in the data. You see imbalanced data a lot in financial transaction data where the majority of the transactions are not fraudulent and a small number of the transactions are fraudulent. Building a predictive model on the imbalanced data set would cause a model that appears to yield high accuracy but does not generalize well to the new data in the minority class. To prevent creating models with false levels of accuracy, you should rebalance your imbalanced data before creating a predictive model.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

The following example shows you how to use the BALANCE function to create a more balanced data set.

1. View the distribution of the classes.

```
=> SELECT fraud, COUNT(fraud) FROM transaction_data GROUP BY fraud;
 fraud | COUNT
-----+-----
 TRUE  |    19
 FALSE |   981
(2 rows)
```

2. Use the BALANCE function to create a more balanced data set.

```
=> SELECT BALANCE('balance_fin_data', 'transaction_data', 'fraud', 'under_sampling'
                USING PARAMETERS sampling_ratio = 0.2);
      BALANCE
-----
Finished in 1 iteration

(1 row)
```

3. View the new distribution of the classifiers.

```
=> SELECT fraud, COUNT(fraud) FROM balance_fin_data GROUP BY fraud;
 fraud | COUNT
-----+-----
 t     |    19
 f     |   236
(2 rows)
```

See Also

- [BALANCE](#)

Detecting Outliers

Before you perform an in-depth analysis of your data, you should first remove the outliers from the data. Outliers are data points that greatly differ from other similar data points. If you leave outliers in your data, then you risk misclassifying data, introducing bias, or incorrect calculations.

For this example we will use the baseball data set found on the Vertica GitHub page.

1. Create a table.

```
=> CREATE TABLE baseball (id identity, first_name varchar(50), last_name varchar(50), dob DATE,
                          team varchar(20), hr int, hits int, avg float, salary float);
```

```
CREATE TABLE
```

2. Load the data.

```
=> COPY baseball FROM 'baseball.csv' DELIMITER ',';
Rows Loaded
-----
      1000
(1 row)
```

3. Detect the outliers based on the hr, hits, and salary columns. We will use the id and team columns as our key columns. The DETECT_OUTLIERS function will create a view containing the outliers with the input and key columns. Before you use the DETECT_OUTLIERS function, make sure that you are a superuser or have CREATE privileges for the schema and SELECT privileges for the table.

```
=> SELECT DETECT_OUTLIERS('baseball_hr_hits_salary_outliers', 'baseball', 'hr, hits, salary',
'robust_zscore'
                        USING PARAMETERS outlier_threshold=3.0, key_columns='id, team');
DETECT_OUTLIERS
-----
Detected 5 outliers
(1 row)
```

4. Query the output view containing the outliers.

```
=> SELECT * FROM baseball_hr_hits_salary_outliers;
id | team | hr | hits | salary
-----+-----+-----+-----+-----
 73 | Mauv | 8888 | 34 | 9.99999999341471e+16
 89 | Pink | 273333 | 4490260 | 4.44444444444828e+17
147 | Maroon | 1100037 | 230 | 9000101403
 87 | Green | 80 | 64253 | 16032567.12
222 | Goldenrod | 3200000 | 216 | 37008899.76
(5 rows)
```

5. Create a view omitting the outliers from the table.

```
=> CREATE VIEW clean_baseball AS
SELECT * FROM baseball WHERE id NOT IN (SELECT id FROM baseball_hr_hits_salary_outliers);
CREATE VIEW
```

6. Perform your analysis.

See Also

- [DETECT_OUTLIERS](#)
- [LINEAR_REG](#)
- [LOGISTIC_REG](#)

Imputing Missing Values

You can use the [IMPUTE](#) function to impute missing values in your data set. The function helps you impute missing values with plausible data values. This impute example uses the `small_input_impute` table. Using the function, you can specify either the mean or mode method.

These examples show how you can use the `IMPUTE` function on the `small_input_impute` table.

Create the Table

First, create the `small_input_impute` table:

```
=> CREATE TABLE small_input_impute(pid int, pclass int, gender int, x1 float, x2 float, x3 float, x4 INT, x5 char, x6 varchar);
```

Then insert the table values:

```
=> INSERT INTO small_input_impute VALUES( 1, 0, 0, -9.445818, -9.740541, -9.786974, 3, 't', 'A');
=> INSERT INTO small_input_impute VALUES( 2, 0, 0, -9.618292, -9.308881, -9.562255, 4, 't', 'A');
=> INSERT INTO small_input_impute VALUES( 3, 0, 0, -9.060605, -9.390844, -9.559848, 6, 't', 'B');
=> INSERT INTO small_input_impute VALUES( 4, 0, 0, -2.264599, -2.615146, -2.107290, 15, 't', 'B');
=> INSERT INTO small_input_impute VALUES( 5, 0, 1, -2.590837, -2.892819, -2.702960, 2, 't', 'C');
=> INSERT INTO small_input_impute VALUES( 6, 0, 1, -2.264599, -2.615146, -2.107290, 11, 't', 'C');
=> INSERT INTO small_input_impute VALUES( 7, 1, 1, 3.829239, 3.087650, 'INFINITY', NULL, 'f', 'C');
=> INSERT INTO small_input_impute VALUES( 8, 1, 1, 3.273592, NULL, 3.477332, 18, 'f', 'B');
=> INSERT INTO small_input_impute VALUES( 9, 1, 1, NULL, 3.841606, 3.754375, 20, 'f', 'B');
=> INSERT INTO small_input_impute VALUES( 10,1, 1, NULL, 3.841606, 3.754375, 20, 't', 'A');
=> INSERT INTO small_input_impute VALUES( 11, 0, 0, -9.445818, -9.740541, -9.786974, 3, 't', 'B');
=> INSERT INTO small_input_impute VALUES( 12, 0, 0, -9.618292, -9.308881, -9.562255, 4, 't', 'C');
=> INSERT INTO small_input_impute VALUES( 13, 0, 0, -9.060605, -9.390844, -9.559848, 6, 't', 'C');
=> INSERT INTO small_input_impute VALUES( 14, 0, 0, -2.264599, -2.615146, -2.107290, 15, 'f', 'A');
=> INSERT INTO small_input_impute VALUES( 15, 0, 1, -2.590837, -2.892819, -2.702960, 2, 'f', 'A');
=> INSERT INTO small_input_impute VALUES( 16, 0, 1, -2.264599, -2.615146, -2.107290, 11, 'f', 'A');
=> INSERT INTO small_input_impute VALUES( 17, 1, 1, 3.829239, 3.087650, 'INFINITY', NULL, 'f',
```

```
'B');
=> INSERT INTO small_input_impute VALUES( 18, 1, 1, 3.273592, NULL, 3.477332, 18, 't', 'B');
=> INSERT INTO small_input_impute VALUES( 19, 1, 1, NULL, 3.841606, 3.754375, 20, 't', NULL);
=> INSERT INTO small_input_impute VALUES( 20,1, 1, NULL, 3.841606, 3.754375, 20, NULL, 'C');
```

You can see the table with the missing values:

```
=> SELECT * FROM small_input_impute;
pid | pclass | gender | x1 | x2 | x3 | x4 | x5 | x6
-----+-----+-----+-----+-----+-----+-----+-----+-----
 1 | 0 | 0 | -9.445818 | -9.740541 | -9.786974 | 3 | t | A
 2 | 0 | 0 | -9.618292 | -9.308881 | -9.562255 | 4 | t | A
 3 | 0 | 0 | -9.060605 | -9.390844 | -9.559848 | 6 | t | B
 4 | 0 | 0 | -2.264599 | -2.615146 | -2.10729 | 15 | t | B
 6 | 0 | 1 | -2.264599 | -2.615146 | -2.10729 | 11 | t | C
 8 | 1 | 1 | 3.273592 | | 3.477332 | 18 | f | B
10 | 1 | 1 | | 3.841606 | 3.754375 | 20 | t | A
18 | 1 | 1 | 3.273592 | | 3.477332 | 18 | t | B
19 | 1 | 1 | | 3.841606 | 3.754375 | 20 | t |
20 | 1 | 1 | | 3.841606 | 3.754375 | 20 | | C
 5 | 0 | 1 | -2.590837 | -2.892819 | -2.70296 | 2 | t | C
 7 | 1 | 1 | 3.829239 | 3.08765 | Infinity | | f | C
13 | 0 | 0 | -9.060605 | -9.390844 | -9.559848 | 6 | t | C
14 | 0 | 0 | -2.264599 | -2.615146 | -2.10729 | 15 | f | A
15 | 0 | 1 | -2.590837 | -2.892819 | -2.70296 | 2 | f | A
16 | 0 | 1 | -2.264599 | -2.615146 | -2.10729 | 11 | f | A
 9 | 1 | 1 | | 3.841606 | 3.754375 | 20 | f | B
11 | 0 | 0 | -9.445818 | -9.740541 | -9.786974 | 3 | t | B
12 | 0 | 0 | -9.618292 | -9.308881 | -9.562255 | 4 | t | C
17 | 1 | 1 | 3.829239 | 3.08765 | Infinity | | f | B
(20 rows)
```

Specify the Mean Method

Execute the IMPUTE function, specifying the mean method, without using the partition_ columns parameter:

```
=> SELECT IMPUTE('output_view','small_input_impute', 'pid, x1,x2,x3,x4','mean'
                USING PARAMETERS exclude_columns='pid');
impute
-----
Finished in 1 iteration
(1 row)
```

View output_view to see the imputed values:

```
=> SELECT * FROM output_view;
x1 | x2 | x3 | x4
-----+-----+-----+-----
-9.445818 | -9.740541 | -9.786974 | 3
```

```

-9.618292 | -9.308881 | -9.562255 | 4
-9.060605 | -9.390844 | -9.559848 | 6
-2.264599 | -2.615146 | -2.10729 | 15
-2.264599 | -2.615146 | -2.10729 | 11
3.273592 | -2.865835 | 3.477332 | 18
-3.517739875 | 3.841606 | 3.754375 | 20
3.273592 | -2.865835 | 3.477332 | 18
-3.517739875 | 3.841606 | 3.754375 | 20
-3.517739875 | 3.841606 | 3.754375 | 20
-3.517739875 | 3.841606 | 3.754375 | 20
-9.445818 | -9.740541 | -9.786974 | 3
-9.618292 | -9.308881 | -9.562255 | 4
3.829239 | 3.08765 | -2.760059444444444 | 11
-2.590837 | -2.892819 | -2.70296 | 2
3.829239 | 3.08765 | -2.760059444444444 | 11
-9.060605 | -9.390844 | -9.559848 | 6
-2.264599 | -2.615146 | -2.10729 | 15
-2.590837 | -2.892819 | -2.70296 | 2
-2.264599 | -2.615146 | -2.10729 | 11
(20 rows)

```

You can also execute the IMPUTE function, specifying the mean method and using the partition_columns parameter. This parameter uses the GROUP BY clause:

```

=> SELECT impute('output_view_group','small_input_impute', 'pid, x1,x2,x3,x4','mean'
USING PARAMETERS exclude_columns='pid', partition_columns='pclass,gender');
impute
-----
Finished in 1 iteration
(1 row)

```

View output_view_group to see the imputed values:

```

=> SELECT * FROM output_view_group;

pclass | gender | x1 | x2 | x3 | x4
-----+-----+-----+-----+-----+-----
0 | 0 | -9.445818 | -9.740541 | -9.786974 | 3
0 | 0 | -9.618292 | -9.308881 | -9.562255 | 4
0 | 0 | -9.060605 | -9.390844 | -9.559848 | 6
0 | 0 | -2.264599 | -2.615146 | -2.10729 | 15
0 | 1 | -2.264599 | -2.615146 | -2.10729 | 11
1 | 1 | 3.273592 | 3.590287333333333 | 3.477332 | 18
1 | 1 | 3.5514155 | 3.841606 | 3.754375 | 20
1 | 1 | 3.273592 | 3.590287333333333 | 3.477332 | 18
1 | 1 | 3.5514155 | 3.841606 | 3.754375 | 20
1 | 1 | 3.5514155 | 3.841606 | 3.754375 | 20
0 | 1 | -2.590837 | -2.892819 | -2.70296 | 2
1 | 1 | 3.829239 | 3.08765 | 3.662027333333333 | 19
0 | 0 | -9.060605 | -9.390844 | -9.559848 | 6
0 | 0 | -2.264599 | -2.615146 | -2.10729 | 15
0 | 1 | -2.590837 | -2.892819 | -2.70296 | 2
0 | 1 | -2.264599 | -2.615146 | -2.10729 | 11
1 | 1 | 3.5514155 | 3.841606 | 3.754375 | 20
0 | 0 | -9.445818 | -9.740541 | -9.786974 | 3

```

```
0      |      0 | -9.618292 |      -9.308881 |      -9.562255 | 4
1      |      1 |  3.829239 |      3.08765  |  3.66202733333333 | 19
(20 rows)
```

Specify the Mode Method

Execute the IMPUTE function, specifying the mode method, without using the `partition_columns` parameter:

```
=> SELECT impute('output_view_mode','small_input_impute', 'pid, x5,x6','mode'
                USING PARAMETERS exclude_columns='pid');
impute
-----
Finished in 1 iteration
(1 row)
```

View `output_view_mode` to see the imputed values:

```
=> SELECT * FROM output_view_mode;

x5 | x6
---+---
t  | A
t  | A
t  | B
t  | B
t  | C
f  | B
t  | A
t  | B
t  | B
t  | C
f  | B
t  | B
t  | C
f  | B
t  | C
f  | C
t  | C
f  | A
f  | A
f  | A
(20 rows)
```

You can also execute the IMPUTE function, specifying the mode method and using the `partition_columns` parameter. This parameter uses the `GROUP BY` clause:

```
=> SELECT impute('output_view_mode_group','small_input_impute', 'pid, x5,x6','mode'
                USING PARAMETERS exclude_columns='pid',partition_columns='pclass,gender');
impute
```

```
-----  
Finished in 1 iteration  
(1 row)
```

View `output_view_mode_group` to see the imputed values:

```
=> SELECT * FROM output_view_mode_group;  
pclass | gender | x5 | x6  
-----+-----+-----+-----  
0      |      | t  | B  
0      |      | t  | C  
0      |      | t  | A  
0      |      | t  | A  
0      |      | t  | B  
0      |      | t  | B  
0      |      | t  | C  
0      |      | f  | A  
0      |      | 1  | C  
0      |      | 1  | C  
0      |      | 1  | A  
0      |      | 1  | A  
1      |      | 1  | C  
1      |      | 1  | B  
1      |      | 1  | A  
1      |      | 1  | B  
1      |      | 1  | B  
1      |      | 1  | C  
1      |      | 1  | B  
1      |      | 1  | B  
(20 rows)
```

See Also

[IMPUTE](#)

Normalizing Data

The purpose of normalization is, primarily, to scale numeric data from different columns down to an equivalent scale. For example, suppose you execute the [LINEAR_REG](#) function on a data set with two feature columns, `current_salary` and `years_worked`. The output value you are trying to predict is a worker's future salary. The values in the `current_salary` column are likely to have a far wider range, and much larger values, than the values in the `years_worked` column. Therefore, the values in the `current_salary` column can overshadow the values in the `years_worked` column, thus skewing your model.

Vertica offers the following data preparation methods which use normalization. These methods are:

- **MinMax**
Using the MinMax normalization method, you can normalize the values in both of these columns to be within a distribution of values between 0 and 1. Doing so allows you to compare values on very different scales to one another by reducing the dominance of one column over the other.
- **Z-score**
Using the Z-score normalization method, you can normalize the values in both of these columns to be the number of standard deviations an observation is from the mean of each column. This allows you to compare your data to a normally distributed random variable.
- **Robust Z-score**
Using the Robust Z-score normalization method, you can lessen the influence of outliers on Z-score calculations. Robust Z-score normalization uses the median value as opposed to the mean value used in Z-score. By using the median instead of the mean, it helps remove some of the influence of outliers in the data.

Normalizing data results in the creation of a view where the normalized data is saved. The `output_view` option in the `NORMALIZE` function determines name of the view .

Normalizing Salary Data Using MinMax

The following example shows how you can normalize the `salary_data` table using the MinMax normalization method.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

```
=> SELECT NORMALIZE('normalized_salary_data', 'salary_data', 'current_salary, years_worked',  
  'minmax');  
      NORMALIZE  
-----  
Finished in 1 iteration  
  
(1 row)  
  
=> SELECT * FROM normalized_salary_data;  
current_salary | years_worked  
-----+-----  
0.437246565765357217 | 0.350000000000000000  
0.978867411144492943 | 0.100000000000000000  
0.909048995710749580 | 0.250000000000000000  
0.601863084103319918 | 0.100000000000000000  
0.455949209228501786 | 0.050000000000000000  
0.538816771536005140 | 0.400000000000000000
```

```
0.183954046444834949 | 0.900000000000000000
0.735279557092379495 | 0.100000000000000000
0.671828883472214349 | 0.800000000000000000
0.092901007123556866 | 0.350000000000000000
0.647827976494151881 | 0.150000000000000000
0.779555202841864596 | 0.600000000000000000
0.942803695935604558 | 0.050000000000000000
0.919445231751790337 | 0.900000000000000000
0.314180979914214992 | 0.800000000000000000
.
.
.
(1000 rows)
```

Normalizing Salary Data Using Z-score

The following example shows how you can normalize the `salary_data` table using the Z-score normalization method.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

```
=> SELECT NORMALIZE('normalized_z_salary_data', 'salary_data', 'current_salary, years_worked',
                    'zscore');
NORMALIZE
-----
Finished in 1 iteration

(1 row)

=> SELECT * FROM normalized_z_salary_data;
employee_id | current_salary | years_worked
-----+-----+-----
189 | -0.221041249770669 | -0.524447274157005
518 | 1.66054215981221 | -1.35743214416495
1126 | 1.41799393943946 | -0.857641222160185
1157 | 0.350834283622416 | -1.35743214416495
1277 | -0.156068522159045 | -1.52402911816654
3188 | 0.131812255991634 | -0.357850300155415
3196 | -1.10097599783475 | 1.30811943986048
3430 | 0.814321286168547 | -1.35743214416495
3522 | 0.593894513770248 | 0.974925491857304
3892 | -1.41729301118583 | -0.524447274157005
3939 | 0.510515691167414 | -1.19083517016336
4165 | 0.968134273983823 | 0.308537595850945
4335 | 1.53525730638774 | -1.52402911816654
4534 | 1.454110321266 | 1.30811943986048
4806 | -0.648569411959509 | 0.974925491857304
.
.
.
(1000 rows)
```

Normalizing Salary Data Using Robust Z-score

The following example shows how you can normalize the `salary_data` table using the Z-score normalization method.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

```
=> SELECT NORMALIZE('normalized_robustz_salary_data', 'salary_data', 'current_salary, years_worked',
'robust_zscore');
      NORMALIZE
-----
Finished in 1 iteration

(1 row)

=> SELECT * FROM normalized_robustz_salary_data;
employee_id | current_salary | years_worked
-----+-----+-----
      110 | -0.568102001091691813 | -1.348981518953190
      112 | -0.459684645251655561 | -0.404694455685957
      313 |  1.192992574435873405 |  0.404694455685957
      426 | -0.339722133536566526 |  0.134898151895319
      593 |  1.006869419240553261 |  1.214083367057871
      620 |  0.488357003820921347 |  0.944287063267233
      630 | -0.381700773453350773 | -0.404694455685957
      718 | -1.218698826946249627 | -0.539592607581276
      777 | -1.184495916454679341 |  0.809388911371914
     1202 | -0.572973418334243077 |  1.214083367057871
     1464 | -0.133042215516442321 |  0.539592607581276
     1573 | -1.056139736552107686 |  0.539592607581276
     1613 | -1.101208070280235229 |  0.539592607581276
     1792 | -0.152599977342245383 | -0.944287063267233
     2433 | -0.469849737890975004 |  0.404694455685957
.
.
.
(1000 rows)
```

See Also

- [NORMALIZE](#)

Sampling Data

The goal of data sampling is to take a smaller, more manageable sample of a much larger data set. With a sample data set, you can produce predictive models or use it to help you tune your

database. The following example shows how you can use the TABLESAMPLE clause to create a sample of your data.

Sampling Data from a Table

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

Using the `baseball` table, create a new table named `baseball_sample` containing a 25% sample of `baseball`. Remember, TABLESAMPLE does not guarantee that the exact percentage of records defined in the clause are returned.

```
=> CREATE TABLE baseball_sample AS SELECT * FROM baseball TABLESAMPLE(25);
CREATE TABLE
=> SELECT * FROM baseball_sample;
id | first_name | last_name | dob      | team   | hr | hits | avg  | salary
-----+-----+-----+-----+-----+---+-----+-----+-----
 4 | Amanda     | Turner   | 1997-12-22 | Maroon | 58 | 177 | 0.187 | 8047721
20 | Jesse     | Cooper   | 1983-04-13 | Yellow | 97 | 39  | 0.523 | 4252837
22 | Randy     | Peterson | 1980-05-28 | Orange | 14 | 16  | 0.141 | 11827728.1
24 | Carol     | Harris   | 1991-04-02 | Fuscia | 96 | 12  | 0.456 | 40572253.6
32 | Rose      | Morrison | 1977-07-26 | Goldenrod | 27 | 153 | 0.442 | 14510752.49
50 | Helen     | Medina   | 1987-12-26 | Maroon | 12 | 150 | 0.54  | 32169267.91
70 | Richard   | Gilbert  | 1983-07-13 | Khaki  | 1  | 250 | 0.213 | 40518422.76
81 | Angela    | Cole     | 1991-08-16 | Violet | 87 | 136 | 0.706 | 42875181.51
82 | Elizabeth | Foster   | 1994-04-30 | Indigo | 46 | 163 | 0.481 | 33896975.53
98 | Philip    | Gardner  | 1992-05-06 | Puce   | 39 | 239 | 0.697 | 20967480.67
102 | Ernest   | Freeman  | 1983-10-05 | Turquoise | 46 | 77  | 0.564 | 21444463.92
.
.
.
(227 rows)
```

With your sample you can create a predictive model, or tune your database.

See Also

- [FROM Clause](#) (for more information about the TABLESAMPLE clause)
- [Building a Linear Regression Model](#)
- [Building a Logistic Regression Model](#)
- [Clustering Data Using k-means](#)

Regression Algorithms

Regression is an important and popular machine learning tool that makes predictions from data by learning the relationship between some features of the data and an observed value response. Regression is used to make predictions about profits, sales, temperature, stocks, and more. For example, you could use regression to predict the price of a house based on the location, the square footage, the size of the lot, and so on. In this example, the house's value is the outcome, and the other factors, such as location, are the features.

The optimal set of coefficients found for the regression's equation is known as the model. The relationship between the outcome and the features is summarized in the model, which can then be applied to different data sets, where the outcome value is unknown. The data in regression projects is divided into two data sets - one that is used to build the model, and one that is used to test the model.

Vertica supports two algorithms for regression:

- [Linear Regression](#)
- [SVM \(Support Vector Machine\) for Regression](#)

Linear Regression

Using linear regression, you can model the linear relationship between independent variables, or features, and a dependent variable, or outcome. You can build linear regression models to:

- Fit a predictive model to a training data set of independent variables and some dependent variable. Doing so allows you to use feature variable values to make predictions on outcomes. For example, you can predict the amount of rain that will fall on a particular day of the year.
- Determine the strength of the relationship between an independent variable and some outcome variable. For example, suppose you want to determine the importance of various weather variables on the outcome of how much rain will fall. You can build a linear regression model based on observations of weather patterns and rainfall to find the answer.

Unlike [Logistic Regression](#), which you use to determine a binary classification outcome, linear regression is primarily used to predict continuous numerical outcomes in linear relationships.

You can use the following functions to build a linear regression model, view the model, and use the model to make predictions on a set of test data:

- [LINEAR_REG](#)
- [PREDICT_LINEAR_REG](#)
- [SUMMARIZE_MODEL](#)

For a complete programming example of how to use linear regression on a table in Vertica, see [Building a Linear Regression Model](#).

Building a Linear Regression Model

This linear regression example uses a small data set named `faithful`. The example shows how you can build a model to predict the value of `eruptions`, given the value of the `waiting` feature.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

1. Create the linear regression model, named `linear_reg_faithful`, using the `faithful_training` training data:

```
=> SELECT LINEAR_REG('linear_reg_faithful', 'faithful_training', 'eruptions', 'waiting'
                    USING PARAMETERS optimizer='BFGS');
      LINEAR_REG
-----
Finished in 6 iterations

(1 row)
```

2. View the summary output of `linear_reg_faithful`:

```
=> SELECT SUMMARIZE_MODEL('linear_reg_faithful');
-[ RECORD 1 ]-----+-----
SUMMARIZE_MODEL | coeff names : {Intercept, waiting}
coefficients:   {-2.067947819, 0.07875927383}
std_err:        {0.21063, 0.0029203}
t_value:        {-9.8178, 26.969}
p_value:        {4.2157e-18, 2.1094e-61}
Number of iterations: 6, Number of skipped samples: 0, Number of processed samples: 162
```

```
Call: linearReg(model_name=linear_reg_faithful, input_table=faithful_training, response_
column=eruptions,
predictor_columns=waiting,
exclude_columns=, optimizer=bfgs, epsilon=0.0001, max_iterations=50)
```

3. Create a new table that contains the response values from running the PREDICT_LINEAR_REG function on your test data. Name this table pred_faithful_results:

```
=> CREATE TABLE pred_faithful_results AS
      (SELECT id, eruptions, PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='linear_
reg_faithful')
      AS pred FROM faithful_testing);
CREATE TABLE
```

4. View the results in the pred_faithful_results table:

```
=> SELECT * FROM pred_faithful_results ORDER BY id;
 id | eruptions |      pred
-----+-----+-----
  4 |    2.283 | 2.8151271587036
  5 |    4.533 | 4.62659045686076
  8 |     3.6 | 4.62659045686076
  9 |     1.95 | 1.94877514654148
 11 |     1.833 | 2.18505296804024
 12 |     3.917 | 4.54783118302784
 14 |     1.75 | 1.6337380512098
 20 |     4.25 | 4.15403481386324
 22 |     1.75 | 1.6337380512098
.
.
.
(110 rows)
```

Calculating the Mean Squared Error (MSE)

Another way that you can calculate how well your model fits the data is by using the MSE function. MSE returns the average of the squared differences between actual value and predicted values.

```
=> SELECT MSE (eruptions::float, pred::float) OVER() FROM
      (SELECT eruptions, pred FROM pred_faithful_results) AS prediction_output;
      mse |      Comments
-----+-----
 0.252925741352641 | Of 110 rows, 110 were used and 0 were ignored
(1 row)
```

See Also

- [LINEAR_REG](#)
- [PREDICT_LINEAR_REG](#)
- [SUMMARIZE_MODEL](#)
- [RSQUARED](#)
- [MSE](#)

SVM (Support Vector Machine) for Regression

Support Vector Machine (SVM) for regression predicts continuous ordered variables based on the training data. This supervised learning method has a number of applications, including:

- Predicting time series
- Pattern recognition
- Function estimation

Unlike [Logistic Regression](#), which you use to determine a binary classification outcome, SVM for regression is primarily used to predict continuous numerical outcomes.

You can use the following functions to build an SVM for regression model, view the model, and use the model to make predictions on a set of test data:

- [SVM_REGRESSOR](#)
- [PREDICT_SVM_REGRESSOR](#)
- [SUMMARIZE_MODEL](#)

For a complete example of how to use the SVM algorithm in Vertica, see [Building an SVM for Regression Model](#) .

Building an SVM for Regression Model

This SVM for regression example uses a small data set named `faithful`, based on the Old Faithful geyser in Yellowstone National Park. The data set contains values about the waiting

time between eruptions and the duration of eruptions of the geyser. The example shows how you can build a model to predict the value of eruptions, given the value of the waiting feature.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

1. Create the SVM model, named `svm_faithful`, using the `faithful_training` training data:

```
=> SELECT SVM_REGRESSOR('svm_faithful', 'faithful_training', 'eruptions', 'waiting'
                        USING PARAMETERS error_tolerance=0.1, max_iterations=100);
SVM_REGRESSOR
-----
Finished in 5 iterations
Accepted Rows: 162   Rejected Rows: 0
(1 row)
```

2. View the summary output of `svm_faithful`:

```
=> SELECT SUMMARIZE_MODEL('svm_faithful');
-----
=====
Predictors and Coefficients
=====
      |Coefficients
-----+-----
Intercept|  -1.59007
waiting  |   0.07217
=====
Notes
=====
Call string:
SELECT svm_regressor('svm_faithful', 'faithful_training', "eruptions", 'waiting'
USING PARAMETERS error_tolerance = 0.1, C=1, max_iterations=100, epsilon=0.001);
Number of rows accepted: 162
Number of rows rejected: 0
Training finished in 5 iterations.
(1 row)
```

3. Create a new table that contains the response values from running the `PREDICT_SVM_REGRESSOR` function on your test data. Name this table `pred_faithful_results`:

```
=> CREATE TABLE pred_faithful AS
      (SELECT id, eruptions, PREDICT_SVM_REGRESSOR(waiting USING PARAMETERS model_name='svm_
faithful')
      AS pred FROM faithful_testing);
CREATE TABLE
```

4. View the results in the `pred_faithful_results` table:

```
=> SELECT * FROM pred_faithful ORDER BY id;
id | eruptions |      pred
-----+-----+-----
 4 |    2.283 | 2.88444568755189
 5 |    4.533 | 4.54434581879796
 8 |     3.6 | 4.54434581879796
 9 |     1.95 | 2.09058040739072
11 |    1.833 | 2.30708912016195
12 |    3.917 | 4.47217624787422
14 |     1.75 | 1.80190212369576
20 |     4.25 | 4.11132839325551
22 |     1.75 | 1.80190212369576
.
.
.
(110 rows)
```

Calculating the Mean Squared Error (MSE)

Another way that you can calculate how well your model fits the data is by using the MSE function. MSE returns the average of the squared differences between actual value and predicted values.

```
=> SELECT MSE(obs::float, prediction::float) OVER()
FROM (SELECT eruptions AS obs, pred AS prediction
      FROM pred_faithful) AS prediction_output;
mse | Comments
-----+-----
0.254499811834235 | Of 110 rows, 110 were used and 0 were ignored
(1 row)
```

See Also

- [SVM \(Support Vector Machine\) for Regression](#)
- [SVM_REGRESSOR](#)
- [PREDICT_SVM_REGRESSOR](#)
- [SUMMARIZE_MODEL](#)

Classification Algorithms

Classification is an important and popular machine learning tool that assigns items in a data set to different categories. Classification is used to predict risk over time, in fraud detection, text categorization, and more. Classification functions begin with a data set where the different

categories are known. For example, suppose you want to classify students based on how likely they are to get into graduate school. In addition to factors like admission score exams and grades, you could also track work experience.

Binary classification means the outcome, in this case, admission, only has two possible values: admit or do not admit. Multiclass outcomes have more than two values. For example, low, medium, or high chance of admission. During the training process, classification algorithms find the relationship between the outcome and the features. This relationship is summarized in the model, which can then be applied to different data sets, where the categories are unknown. The data in classification projects is divided into two data sets - one that is used to build the model, and one that is used to test the model.

Vertica supports four algorithms for classification:

- [Logistic Regression](#)
- [Naive Bayes](#)
- [Random Forest](#)
- [SVM \(Support Vector Machine\) for Classification](#)

Logistic Regression

Using logistic regression, you can model the relationship between independent variables, or features, and some dependent variable, or outcome. The outcome of logistic regression is always a binary value.

You can build logistic regression models to:

- Fit a predictive model to a training data set of independent variables and some binary dependent variable. Doing so allows you to make predictions on outcomes, such as whether a piece of email is spam mail or not.
- Determine the strength of the relationship between an independent variable and some binary outcome variable. For example, suppose you want to determine whether an email is spam or not. You can build a logistic regression model, based on observations of the properties of email messages. Then, you can determine the importance of various properties of an email message on that outcome.

You can use the following functions to build a logistic regression model, view the model, and use the model to make predictions on a set of test data:

- [LOGISTIC_REG](#)
- [PREDICT_LOGISTIC_REG](#)
- [SUMMARIZE_MODEL](#)

For a complete programming example of how to use logistic regression on a table in Vertica, see [Building a Logistic Regression Model](#).

Building a Logistic Regression Model

This logistic regression example uses a small data set named `mtcars`. The example shows you how to build a model to predict the value of `am` (whether the car has an automatic or a manual transmission). It uses the given values of all the other features in the data set.

In this example, roughly 60% of the data is used as training data to create a model. The remaining 40% of the data is used as testing data against which you can test your logistic regression model.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

1. Create the logistic regression model, named `logistic_reg_mtcars`, using the `mtcars_train` training data.

```
=> SELECT LOGISTIC_REG('logistic_reg_mtcars', 'mtcars_train', 'am', 'cyl, hp, wt'
                        USING PARAMETERS exclude_columns='hp');
LOGISTIC_REG
-----
Finished in 15 iterations

(1 row)
```

2. View the summary output of `logistic_reg_mtcars`.

```
=> SELECT SUMMARIZE_MODEL('logistic_reg_mtcars');
-[ RECORD 1 ]-----+-----
coeff names : {Intercept, cyl, wt}
coefficients: {174.4571914, 5.878990016, -70.06270163}
std_err:     {1236.6, 46.252, 500.04}
z_value:     {0.14108, 0.12711, -0.14011}
p_value:     {0.88781, 0.89886, 0.88857}
Number of iterations: 15, Number of skipped samples: 0, Number of processed samples: 20
Call: logisticReg(model_name=logistic_reg_mtcars, input_table=mtcars_train, response_column=am,
predictor_columns=cyl, hp, wt,
```

```
exclude_columns='hp', optimizer=bfgs, epsilon=0.0001, max_iterations=100)
```

3. Create a new table, named `mtcars_predict_results`. Populate this table with the prediction outputs you obtain from running the `PREDICT_LOGISTIC_REG` function on your test data.

```
=> CREATE TABLE mtcars_predict_results AS
      (SELECT car_model, am, PREDICT_LOGISTIC_REG(cyl, hp, wt
                                                USING PARAMETERS model_name='logistic_reg_mtcars')
      AS Prediction FROM mtcars_test);

CREATE TABLE
```

4. View the results in the `mtcars_predict_results` table.

```
=> SELECT * FROM mtcars_predict_results;
  car_model | am | Prediction
-----+---+-----
AMC Javelin | 0 |          0
Hornet 4 Drive | 0 |          0
Maserati Bora | 1 |          0
Merc 280 | 0 |          0
Merc 450SL | 0 |          0
Toyota Corona | 0 |          1
Volvo 142E | 1 |          1
Camaro Z28 | 0 |          0
Datsun 710 | 1 |          1
Honda Civic | 1 |          1
Porsche 914-2 | 1 |          1
Valiant | 0 |          0
(12 rows)
```

5. Evaluate the accuracy of the `PREDICT_LOGISTIC_REG` function, using the [CONFUSION_MATRIX](#) evaluation function.

```
=> SELECT CONFUSION_MATRIX(obs::int, pred::int USING PARAMETERS num_classes=2) OVER()
      FROM (SELECT am AS obs, Prediction AS pred FROM mtcars_predict_results) AS prediction_
output;
  class | 0 | 1 |
-----+---+---+-----
      0 | 6 | 1 |
      1 | 1 | 4 | Of 12 rows, 12 were used and 0 were ignored
(2 rows)
```

In this case, `PREDICT_LOGISTIC_REG` correctly predicted that all five of the cars with a value of 1 in the `am` column have a value of 1. Out of the seven cars which had a value of 0 in the `am` column, six were correctly predicted to have the value 0. One car was incorrectly classified as having the value 1.

See Also

- [CONFUSION_MATRIX](#)
- [LIFT_TABLE](#)
- [LOGISTIC_REG](#)
- [PREDICT_LOGISTIC_REG](#)
- [SUMMARIZE_MODEL](#)

Naive Bayes

You can use the Naive Bayes algorithm to classify your data when features can be assumed independent. The algorithm uses independent features to calculate the probability of a specific class. For example, you might want to predict the probability that an email is spam. In that case, you would use a corpus of words associated with spam to calculate the probability the email's content is spam.

This supervised machine learning algorithm has a number of applications, including:

- Spam filtering
- Classifying documents
- Image classification

You can use the following functions to build a Naive Bayes model, view the model, and use the model to make predictions on a set of test data:

- [NAIVE_BAYES](#)
- [PREDICT_NAIVE_BAYES](#)
- [PREDICT_NAIVE_BAYES_CLASSES](#)
- [SUMMARIZE_MODEL](#)

For a complete example of how to use the Naive Bayes algorithm in Vertica, see [Classifying Data Using Naive Bayes](#).

Classifying Data Using Naive Bayes

This Naive Bayes example uses the HouseVotes84 data set to show you how to build a model. With this model, you can predict which party the member of the United States Congress is affiliated based on their voting record. To aid in classifying the data it has been cleaned, and any missed votes have been replaced. The cleaned data replaces missed votes with the voter's party majority vote. For example, suppose a member of the Democrats had a missing value for vote1 and majority of the Democrats voted in favor. This example replaces all missing Democrats' votes for vote1 with a vote in favor.

In this example, approximately 75% of the cleaned HouseVotes84 data is randomly selected and copied to a training table. The remaining cleaned HouseVotes84 data is used as a testing table.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

1. Create the Naive Bayes model, named `naive_house84_model`, using the `house84_train` training data.

```
=> SELECT NAIVE_BAYES('naive_house84_model', 'house84_train', 'party',
                    '*' USING PARAMETERS exclude_columns='party, id');
                    NAIVE_BAYES
-----
Finished. Accepted Rows: 315 Rejected Rows: 0
(1 row)
```

2. View the summary output of `naive_house84_model`.

```
=> SELECT SUMMARIZE_MODEL('naive_house84_model');
                    SUMMARIZE_MODEL
-----

=====
Classes and Prior Probability
=====
| democrat | republican
-----+-----+-----
Prior | 0.59816 | 0.40184

=====
List of Columns
=====
| Column Index | Model Type
-----+-----+-----
```

```
party |      0      | LABEL
vote1 |      1      | Categorical
vote2 |      2      | Categorical
vote3 |      3      | Categorical
vote4 |      4      | Categorical
vote5 |      5      | Categorical
vote6 |      6      | Categorical
vote7 |      7      | Categorical
vote8 |      8      | Categorical
vote9 |      9      | Categorical
vote10|     10      | Categorical
vote11|     11      | Categorical
vote12|     12      | Categorical
vote13|     13      | Categorical
vote14|     14      | Categorical
vote15|     15      | Categorical
vote16|     16      | Categorical
```

```
=====  
Columns with Categorical Models  
=====  
Conditional Probabilities for Column vote1 (1)  
|democrat|republican  
+-----+-----  
n|0.40102 | 0.80451  
y|0.59898 | 0.19549  
  
Conditional Probabilities for Column vote2 (2)  
|democrat|republican  
+-----+-----  
n|0.44670 | 0.36090  
y|0.55330 | 0.63910  
  
. . .  
(1 row)
```

3. Create a new table, named `predicted_party_naive`. Populate this table with the prediction outputs you obtain from the `PREDICT_NAIVE_BAYES` function on your test data.

```
=> CREATE TABLE predicted_party_naive  
AS SELECT party,  
    PREDICT_NAIVE_BAYES (vote1, vote2, vote3, vote4, vote5,  
        vote6, vote7, vote8, vote9, vote10,  
        vote11, vote12, vote13, vote14,  
        vote15, vote16  
        USING PARAMETERS model_name = 'naive_house84_model',  
        type = 'response') AS Predicted_Party  
FROM house84_test;  
CREATE TABLE
```

4. Calculate the accuracy of the model's predictions.

```
=> SELECT (Predictions.Num_Correct_Predictions / Count.Total_Count) AS Percent_Accuracy
      FROM ( SELECT COUNT(Predicted_Party) AS Num_Correct_Predictions
            FROM predicted_party_naive
            WHERE party = Predicted_Party
          ) AS Predictions,
      ( SELECT COUNT(party) AS Total_Count
        FROM predicted_party_naive
      ) AS Count;
Percent_Accuracy
-----
0.9333333333333333
(1 row)
```

The model correctly predicted the party of the members of Congress based on their voting patterns with 93% accuracy.

Viewing the Probability of Each Class

You can also view the probability of each class. Use `PREDICT_NAIVE_BAYES_CLASSES` to see the probability of each class.

```
=> SELECT PREDICT_NAIVE_BAYES_CLASSES (id, vote1, vote2, vote3, vote4, vote5,
                                       vote6, vote7, vote8, vote9, vote10,
                                       vote11, vote12, vote13, vote14,
                                       vote15, vote16
                                       USING PARAMETERS model_name = 'naive_house84_model',
                                                         key_columns = 'id', exclude_columns = 'id',
                                                         classes = 'democrat, republican')
      OVER() FROM house84_test;
```

id	Predicted	Probability	democrat	republican
368	democrat	1	1	0
372	democrat	1	1	0
374	democrat	1	1	0
378	republican	0.999999962214987	3.77850125111219e-08	0.999999962214987
384	democrat	1	1	0
387	democrat	1	1	0
406	republican	0.999999945980143	5.40198564592332e-08	0.999999945980143
419	democrat	1	1	0
421	republican	0.922808855631005	0.0771911443689949	0.922808855631005
.				
.				
.				

(109 rows)

See Also

- [NAIVE_BAYES](#)
- [PREDICT_NAIVE_BAYES](#)

- [PREDICT_NAIVE_BAYES_CLASSES](#)

Random Forest

The Random Forest algorithm creates an ensemble model of decision trees. Each tree is trained on a randomly selected subset of the training data. This supervised learning method has a number of applications, including:

- Predicting genetic outcomes
- Financial analysis
- Medical diagnosis

You can use the following functions to train the Random Forest model, and use the model to make predictions on a set of test data:

- [RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER_CLASSES](#)
- [SUMMARIZE_MODEL](#)

For a complete example of how to use the Random Forest algorithm in Vertica, see [Classifying Data Using Random Forest](#).

Classifying Data Using Random Forest

This random forest example uses a data set named iris. The example contains four variables that measure various parts of the iris flower to predict its species.

Before you begin the example, make sure that you have followed the steps in [Downloading the Machine Learning Example Data](#).

1. Create the random forest model, named `rf_iris`, using the `iris` data.

```
=> SELECT RF_CLASSIFIER ('rf_iris', 'iris', 'Species', 'Sepal_Length, Sepal_Width, Petal_Length,
Petal_Width'
USING PARAMETERS ntree=100, sampling_size=0.5);

      RF_CLASSIFIER
-----
The random forest is trained

(1 row)
```

2. View the summary output of `rf_iris`.

```
=> SELECT SUMMARIZE_MODEL('rf_iris');
Number of trees: 100
Number of skipped samples: 0, Number of processed samples: 150
Call string:
SELECT rf_classifier('rf_iris', 'iris', '"species"', 'Sepal_Length, Sepal_Width, Petal_Length,
Petal_Width'
USING PARAMETERS exclude_columns='', ntree=100, mtry='2', sampling_size=0.5, max_depth=5, max_
breadth=32, min_leaf_size=1, min_info_gain=0, nbins=32);
Predictor names and types:
sepal_length: float, sepal_width: float, petal_length: float, petal_width: float
(1 row)
```

3. Apply the classifier to the test data:

```
=> SELECT PREDICT_RF_CLASSIFIER (Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
                                USING PARAMETERS model_name='rf_iris') FROM iris1;

PREDICT_RF_CLASSIFIER
-----
setosa
setosa
setosa
.
.
.
versicolor
versicolor
versicolor
.
.
.
virginica
virginica
virginica
.
.
.
(90 rows)
```

4. Use `PREDICT_RF_CLASSES` to view the probability of the classes:

```
=> SELECT PREDICT_RF_CLASSIFIER_CLASSES(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
      USING PARAMETERS model_name='rf_iris') OVER () FROM iris1;
predicted | probability
-----+-----
setosa    |             1
setosa    |             0.99
.
.
.
(90 rows)
```

See Also

- [RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER_CLASSES](#)

SVM (Support Vector Machine) for Classification

Support Vector Machine (SVM) is a classification algorithm that assigns data to one category or the other based on the training data. This algorithm implements linear SVM, which is highly scalable. This supervised learning method has a number of applications, including:

- Image classification
- Text categorization
- Handwritten digit identification

You can use the following functions to train the SVM model, and use the model to make predictions on a set of test data:

- [SVM_CLASSIFIER](#)
- [PREDICT_SVM_CLASSIFIER](#)
- [SUMMARIZE_MODEL](#)

You can also use the following evaluation functions to gain further insights:

- [CONFUSION_MATRIX](#)
- [DETECT_OUTLIERS](#)
- [ERROR_RATE](#)
- [ROC](#)

For a complete example of how to use the SVM algorithm in Vertica, see [Classifying Data Using SVM \(Support Vector Machine\)](#).

The implementation of the SVM algorithm in Vertica is based on the paper [Distributed Newton Methods for Regularized Logistic Regression](#).

Classifying Data Using SVM (Support Vector Machine)

This SVM example uses a small data set named `mtcars`. The example shows how you can use the `SVM_CLASSIFIER` function to train the model to predict the value of `am` (the transmission type, where 0 = automatic and 1 = manual) using the `PREDICT_SVM_CLASSIFIER` function.

Before you begin the example, make sure that you have [Downloading the Machine Learning Example Data](#).

1. Create the SVM model, named `svm_class`, using the `mtcars_train` training data.

```
=> SELECT SVM_CLASSIFIER('svm_class', 'mtcars_train', 'am', 'cyl, mpg, wt, hp, gear'
                          USING PARAMETERS exclude_columns='gear');

SVM_CLASSIFIER
-----
Finished in 12 iterations.
Accepted Rows: 20 Rejected Rows: 0
(1 row)
```

2. View the summary output of `svm_class`.

```
=> SELECT SUMMARIZE_MODEL('svm_class');
-----
=====
Predictors and Coefficients
=====
      |Coefficients
-----+-----
Intercept| -0.02006
cyl      |  0.15367
mpg      |  0.15698
wt       | -1.78157
hp       |  0.00957
=====
Notes
=====
Call string:
SELECT svm_classifier('svm_class', 'mtcars_train', '"am"', 'cyl, mpg, wt, hp, gear'
USING PARAMETERS exclude columns = 'gear', C=1, max_iterations=100, epsilon=0.001);
Number of rows accepted: 20
Number of rows rejected: 0
Training finished in 12 iterations.
(1 row)
```

3. Create a new table, named `svm_mtcars_predict`. Populate this table with the prediction outputs you obtain from running the `PREDICT_SVM_CLASSIFIER` function on your test data.

```
=> CREATE TABLE svm_mtcars_predict AS
      (SELECT car_model, am, PREDICT_SVM_CLASSIFIER(cyl, mpg, wt, hp
                                                    USING PARAMETERS model_name='svm_class')
      AS Prediction FROM mtcars_test);

CREATE TABLE
```

4. View the results in the `svm_mtcars_predict` table.

```
=> SELECT * FROM svm_mtcars_predict;
car_model      | am | Prediction
-----+-----
Toyota Corona |  0 |          1
Camaro Z28     |  0 |          0
Datsun 710     |  1 |          1
Valiant        |  0 |          0
Volvo 142E     |  1 |          1
AMC Javelin    |  0 |          0
Honda Civic    |  1 |          1
Hornet 4 Drive|  0 |          0
Maserati Bora  |  1 |          1
Merc 280       |  0 |          0
Merc 450SL     |  0 |          0
Porsche 914-2 |  1 |          1
(12 rows)
```

5. Evaluate the accuracy of the `PREDICT_SVM_CLASSIFIER` function, using the `CONFUSION_MATRIX` evaluation function.

```
=> SELECT CONFUSION_MATRIX(obs::int, pred::int USING PARAMETERS num_classes=2) OVER()  
      FROM (SELECT am AS obs, Prediction AS pred FROM svm_mtcars_predict1) AS prediction_  
output;  
  class | 0 | 1 |  
-----+-----+-----  
      0 | 6 | 1 |  
      1 | 0 | 5 | Of 12 rows, 12 were used and 0 were ignored  
(2 rows)
```

In this case, `PREDICT_SVM_CLASSIFIER` correctly predicted that the cars with a value of 1 in the `am` column have a value of 1. No cars were incorrectly classified. Out of the seven cars which had a value of 0 in the `am` column, six were correctly predicted to have the value 0. One car was incorrectly classified as having the value 1.

See Also

- [SVM \(Support Vector Machine\) for Classification](#)
- [SVM_CLASSIFIER](#)
- [PREDICT_SVM_CLASSIFIER](#)

Clustering Algorithms

Clustering is an important and popular machine learning tool used to find clusters of items in a data set that are similar to one another. The goal of clustering is to create clusters with a high number of objects that are similar. Similar to classification, clustering segments the data. However, in clustering, the categorical groups are not defined. Clustering can be used to find anomalies in data and find natural groups of data. For example, you can use clustering to analyze a region and determine which areas of that region are most likely to be hit by an earthquake.

In Vertica, clustering is computed based on distance. Through this computation, data points are assigned to the cluster with the nearest mean.

Vertica supports one algorithm for clustering:

- [k-means](#)

k-means

You can use the clustering algorithm, *k-means clustering*, to cluster data points into k different groups based on similarities between the data points. This unsupervised machine learning algorithm has a wide number of applications, including:

- Search engines
- Finding groups of similar customers based on characteristics
- Spam detection
- Cybersecurity

The purpose of k-means is to partition n observations into k clusters. Through this partitioning, k-means assigns each observation to the cluster with the nearest mean. That nearest mean is also known as the *cluster center*.

For a complete programming example of how to use k-means on a table in Vertica, see [Clustering Data Using k-means](#).

Clustering Data Using k-means

This k-means example uses two small data sets named `agar_dish_1` and `agar_dish_2`. Using the numeric data in the `agar_dish_1` data set, you can cluster the data into k clusters. Then, using the created k-means model, you can run `APPLY_KMEANS` on `agar_dish_2` and assign them to the clusters created in your original model.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

Clustering Training Data into k Clusters

1. Using the `KMEANS` function, run k-means on the `agar_dish_1` table.

```
=> SELECT KMEANS('agar_dish_kmeans', 'agar_dish_1', '*', 5
                USING PARAMETERS exclude_columns='id', max_iterations=20, output_view=agar_1_
view,
                key_columns='id');
                KMEANS
```

```
-----  
Finished in 7 iterations  
  
(1 row)
```

The example creates a model named `agar_dish_kmeans` and a view containing the results of the model named `agar_1_view`. You might get different results when you run the clustering algorithm. This is because KMEANS randomly picks initial centers by default.

2. View the output of `agar_1_view`.

```
=> SELECT * FROM agar_1_view;  
id | cluster_id  
-----+-----  
2 | 4  
5 | 4  
7 | 4  
9 | 4  
13 | 4  
.  
.  
.  
(375 rows)
```

3. Because you specified the number of clusters as 5, verify that the function created five clusters. Count the number of data points within each cluster.

```
=> SELECT cluster_id, COUNT(cluster_id) as Total_count  
FROM agar_1_view  
GROUP BY cluster_id;  
cluster_id | Total_count  
-----+-----  
0 | 76  
2 | 80  
1 | 74  
3 | 73  
4 | 72  
(5 rows)
```

From the output, you can see that five clusters were created: 0, 1, 2, 3, and 4.

You have now successfully clustered the data from `agar_dish_1.csv` into five distinct clusters.

Summarizing Your Model

You can also view a summary of the model you created using the [SUMMARIZE_MODEL](#) function. This summary tells you how many cluster centers your model contains, along with other metrics.

```
=> SELECT SUMMARIZE_MODEL('agar_dish_kmeans');
-[ RECORD 1 ]-----+-----
SUMMARIZE_MODEL | k-Means Model Summary:

Number of clusters: 5
Input columns: x, y
Cluster centers:
  0: {x: -7.4811859, y: -7.5257672}
  1: {x: -3.5061558, y: -3.5570295}
  2: {x: -5.5205715, y: -5.4919726}
  3: {x: -1.5623823, y: -1.5056116}
  4: {x:  0.4970753, y:  0.5111612}
Evaluation metrics:
  Total Sum of Squares: 6008.4619
  Within-Cluster Sum of Squares:
    Cluster 0: 12.389038
    Cluster 1: 11.210146
    Cluster 2: 12.994356
    Cluster 3: 12.639238
    Cluster 4: 12.083548
  Total Within-Cluster Sum of Squares: 61.316326
  Between-Cluster Sum of Squares: 5947.1456
  Between-Cluster SS / Total SS: 98.98%
Number of iterations performed: 6
Converged: True
Call:
kmeans(model_name=agar_dish_kmeans, input_table=agar_dish_training, input_columns=*, num_clusters=5,
exclude_columns=id, max_iterations=20, epsilon=0.0001, init_method=random, initial_centers_table=,
distance_method=euclidean, outputView=agar_training_view, key_columns=id
)
```

Clustering Data Using a k-means Model

Using `agar_dish_kmeans`, the k-means model you just created, you can classify the `agar_dish_2` data set.

Create a table named `kmeans_results`, using the `agar_dish_2` table as your input table and the `agar_dish_kmeans` model for your initial cluster centers.

Add only the relevant feature columns to the arguments in the `APPLY_KMEANS` function.

```
=> CREATE TABLE kmeans_results AS
      (SELECT id,
           APPLY_KMEANS(x, y
                       USING PARAMETERS
                           model_name='agar_dish_kmeans') AS cluster_id
      FROM agar_dish_2);
```

The `kmeans_results` table shows that the `agar_dish_kmeans` model correctly clustered the `agar_dish_2` data.

See Also

- [APPLY_KMEANS](#)
- [KMEANS](#)
- [SUMMARIZE_MODEL](#)

Model Management

The following topics explain how to manage your models:

- [Altering Models](#)
- [Drop Models](#)
- [Managing Model Security](#)
- [Summarizing Models](#)
- [Viewing Models](#)

As of 8.0.x, Vertica supports upgrading models.

Altering Models

After you create a model, you can alter it in three ways: renaming the model, changing the owner, and changing the schema.

The following example shows how you can use `ALTER_MODEL` to rename a model:

1. Find the model you want to alter.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mymodel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273816618  
model_name    | mymodel  
schema_id    | 45035996273704978  
schema_name   | public  
owner_id     | 45035996273704962  
owner_name    | dbadmin
```

```
category      | VERTICA_MODELS  
model_type    | kmeans  
is_complete   | t  
create_time   | 2017-03-02 11:16:04.990626-05  
size          | 964
```

2. Rename the model.

```
=> ALTER MODEL mymodel RENAME TO mykmeansmodel;  
ALTER MODEL
```

3. Review V_CATALOG.MODELS to verify that the model name was changed.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273816618  
model_name    | mykmeansmodel  
schema_id     | 45035996273704978  
schema_name   | public  
owner_id      | 45035996273704962  
owner_name    | dbadmin  
category      | VERTICA_MODELS  
model_type    | kmeans  
is_complete   | t  
create_time   | 2017-03-02 11:16:04.990626-05  
size          | 964
```

The following example shows how you can use ALTER_MODEL to change the model owner:

1. Find the model you want to alter.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273816618  
model_name    | mykmeansmodel  
schema_id     | 45035996273704978  
schema_name   | public  
owner_id      | 45035996273704962  
owner_name    | dbadmin  
category      | VERTICA_MODELS  
model_type    | kmeans  
is_complete   | t  
create_time   | 2017-03-02 11:16:04.990626-05  
size          | 964
```

2. Change the model owner.

```
=> ALTER MODEL mykmeansmodel OWNER TO user1;  
ALTER MODEL
```

3. Review V_CATALOG.MODELS to verify that the owner was changed.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273816618  
model_name    | mykmeansmodel  
schema_id     | 45035996273704978  
schema_name   | public  
owner_id      | 45035996273704962  
owner_name    | user1  
category      | VERTICA_MODELS  
model_type    | kmeans  
is_complete   | t  
create_time   | 2017-03-02 11:16:04.990626-05  
size          | 964
```

The following example shows how you can use ALTER_MODEL to change the model schema:

1. Find the model you want to alter.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273816618  
model_name    | mykmeansmodel  
schema_id     | 45035996273704978  
schema_name   | public  
owner_id      | 45035996273704962  
owner_name    | dbadmin  
category      | VERTICA_MODELS  
model_type    | kmeans  
is_complete   | t  
create_time   | 2017-03-02 11:16:04.990626-05  
size          | 964
```

2. Change the model schema.

```
=> ALTER MODEL mykmeansmodel SET SCHEMA test;  
ALTER MODEL
```

3. Review V_CATALOG.MODELS to verify that the owner was changed.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273816618  
model_name    | mykmeansmodel  
schema_id     | 45035996273704978  
schema_name   | test  
owner_id      | 45035996273704962  
owner_name    | dbadmin  
category      | VERTICA_MODELS  
model_type    | kmeans  
is_complete   | t  
create_time   | 2017-03-02 11:16:04.990626-05  
size          | 964
```

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

See Also

- [ALTER MODEL](#)

Drop Models

If you want to remove a model from the database, then you can drop it. To drop a model you must either be the model owner or the dbadmin.

1. Find the model you want to drop.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mySvmClassModel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273765414  
model_name    | mySvmClassModel  
schema_id     | 45035996273704978  
schema_name   | public  
owner_id      | 45035996273704962  
owner_name    | dbadmin  
category      | VERTICA_MODELS  
model_type    | SVM_CLASSIFIER  
is_complete   | t  
create_time   | 2017-02-14 10:30:44.903946-05  
size          | 525
```

2. Drop the model.

```
=> DROP MODEL mySvmClassModel;  
DROP MODEL
```

3. Review V_CATALOG.MODELS to verify that the model was dropped.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mySvmClassModel';  
(0 rows)
```

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

See Also

- [DROP MODEL](#)

Managing Model Security

You can manage the security privileges on your models by using the GRANT and REVOKE statements. The following examples show how you can change privileges on user1 and user2 using the faithful table using the linearReg model.

- In the following example, the dbadmin grants the SELECT privilege to user1:

```
=> GRANT SELECT ON TABLE faithful TO user1;  
GRANT PRIVILEGE
```

- Then, the dbadmin grants the CREATE privilege on the public schema to user1:

```
=> GRANT CREATE ON SCHEMA public TO user1;  
GRANT PRIVILEGE
```

- Connect to the database as user1:

```
=> \c - user1
```

- As user1, build the linearReg model:

```
=> SELECT LINEAR_REG('linearReg', 'faithful', 'waiting', 'eruptions');  
LINEAR_REG  
-----  
Finished in 1 iterations  
(1 row)
```

- As user1, grant USAGE privileges to user2:

```
=> GRANT USAGE ON MODEL linearReg TO user2;  
GRANT PRIVILEGE
```

- Connect to the database as user2:

```
=> \c - user2
```

- To confirm privileges were granted to user2, run the SUMMARIZE_MODEL function. A user with the USAGE privilege on a model can run SUMMARIZE_MODEL on that model:

```
=> SELECT SUMMARIZE_MODEL('linearReg');

summarize_model
-----
-----
-----
-----
-----
coeff names : {Intercept, eruptions}
coefficients: {33.47439702, 10.7296414}
std_err:      {1.155, 0.3148}
t_value:      {28.99, 34.09}
p_value:      {< 1e-20, < 1e-20}
Number of iterations: 1, Number of skipped samples: 0, Number of processed samples: 272
Call:linear_reg('linearReg', 'faithful', '"waiting"', 'eruptions'
USING PARAMETERS optimizer='newton', epsilon=1e-06, max_iterations=100)
(1 row)
```

- Connect to the database as user1:

```
=> \c - user1
```

- Then, you can use the REVOKE statement to revoke privileges from user2:

```
=> REVOKE USAGE ON MODEL linearReg FROM user2;
REVOKE PRIVILEGE
```

- To confirm the privileges were revoked, connect as user 2 and run the SUMMARIZE_MODEL function:

```
=> \c - user2
=>SELECT SUMMARIZE_MODEL('linearReg');
ERROR 7523: Problem in summarize_model.
Detail: Permission denied for model linearReg
```

See Also

- [GRANT \(Model\)](#)
- [REVOKE \(Model\)](#)

Summarizing Models

1. Find the model you want to summarize.

```
=> SELECT * FROM v_catalog.models WHERE model_name='mySvmClassModel';  
-[ RECORD 1 ]-----  
model_id      | 45035996273765414  
model_name    | mySvmClassModel  
schema_id     | 45035996273704978  
schema_name   | public  
owner_id      | 45035996273704962  
owner_name    | dbadmin  
category      | VERTICA_MODELS  
model_type    | SVM_CLASSIFIER  
is_complete   | t  
create_time   | 2017-02-14 10:30:44.903946-05  
size          | 525
```

2. View the model summary.

```
=> SELECT SUMMARIZE_MODEL('mySvmClassModel');  
-[ RECORD 1 ]-----  
-  
SUMMARIZE_MODEL  
  
=====
```

Predictors and Coefficients

```
=====
```

	Coefficients
mpg	0.01116
cyl	0.00078
disp	-0.00074
hp	0.01128
drat	0.00165
wt	0.00037
qsec	0.00626
vs	0.00042
am	0.00047
carb	0.00047
	0.00033

```
=====
```

Notes

```
=====
```

```
Call string:
SELECT svm_classifier('mySvmClassModel', 'mtcars', '"gear"',
'mpg,cyl,disp,hp,drat,wt,qsec,vs,am,carb'
USING PARAMETERS optimizer='TRON', regularization='L2', cost='L2-SVM', C=1, max_iterations=100,
epsilon=0.001);
Number of rows accepted: 32
Number of rows rejected: 0
(1 row)
```

See Also

- [SUMMARIZE_MODEL](#)

Viewing Models

Vertica stores the models you create in the `V_CATALOG.MODELS` system table.

You can query `V_CATALOG.MODELS` to view information about the models you have created:

```
=> SELECT * FROM V_CATALOG.MODELS;
-[ RECORD 1 ]-----
model_id      | 45035996273765414
model_name    | mySvmClassModel
schema_id     | 45035996273704978
schema_name   | public
owner_id      | 45035996273704962
owner_name    | dbadmin
category      | VERTICA_MODELS
model_type    | SVM_CLASSIFIER
is_complete   | t
create_time   | 2017-02-14 10:30:44.903946-05
size          | 525
-[ RECORD 2 ]-----

model_id      | 45035996273711466
model_name    | mtcars_normfit
schema_id     | 45035996273704978
schema_name   | public
owner_id      | 45035996273704962
owner_name    | dbadmin
category      | VERTICA_MODELS
model_type    | SVM_CLASSIFIER
is_complete   | t
create_time   | 2017-02-06 15:03:05.651941-05
size          | 288
```

See Also

- [MODELS](#)

Geospatial Analytics

Vertica provides functions that allows you to manipulate complex two- and three-dimensional spatial objects. These functions follow the Open Geospatial Consortium (OGC) standards. Vertica also provides data types and SQL functions that allow you to specify and store spatial objects in a database according to OGC standards.

Convert Well-Known Text (WKT) and Well-Known Binary (WKB)

Convert WKT and WKB.

Optimized Spatial Joins

Perform fast spatial joins using `ST_Intersects` and `STV_Intersects`.

Load and Export Spatial Data From Shapefiles

Easily load and export shapefiles.

Store and Retrieve Objects

Determine if:

- An object contains self-intersection or self-tangency points.
- One object is entirely within another object, such as a point within a polygon.

Test the relationships between objects

For example, if they intersect or touch:

- Identify the boundary of an object.
- Identify vertices of an object.

Calculate

- Shortest distance between two objects.
- Size of an object (length, area).
- Centroid for one or more objects.
- Buffer around one or more objects.

Best Practices for Geospatial Analytics

Micro Focus recommends the following best practices when performing geospatial analytics in Vertica.

Performance Optimization

Recommendation	Details
Use the minimum column size for spatial data.	Performance degrades as column widths increase. When creating columns for your spatial data, use the smallest size column that can accommodate your data. For example, use GEOMETRY(85) for point data.
Use GEOMETRY types where possible.	Performance of functions on GEOGRAPHY types is slower than functions that support GEOMETRY types. Use GEOMETRY types where possible.
To improve the performance of the following functions, sort projections on spatial columns:	You may improve the pruning efficiency of these functions by sorting the projection on the GEOMETRY column. However, sorting on a large GEOMETRY column may slow down data load.

Recommendation	Details
<ul style="list-style-type: none"> • STV_Intersect scalar function • ST_Distance • ST_Area • ST_Length 	

Spatial Joins with Points and Polygons

Vertica provides two ways to identify whether a set of points intersect with a set of polygons. Depending on the size of your data set, choose the approach that gives the best performance.

For a detailed example of best practices with spatial joins, see [Best Practices for Spatial Joins](#).

Recommendation	Details
Create a spatial index only when performing spatial joins with STV_Intersect.	Spatial indexes should only be used with STV_Intersect. Creating a spatial index and then performing spatial joins with ST_Intersects will not improve performance.
Use the STV_Intersect function when you intersect a set of points with a set of polygons.	Determine if a set of points intersects with a set of polygons in a medium to large data set. First, create a spatial index using STV_Create_Index. Then, use one of the STV_Intersect functions to return the set of pairs that intersect. Spatial indexes provide the best performance for accessing a large number of polygons.
When using the STV_Intersect transform function, partition the data and use an OVER(PARTITION BEST) clause.	The STV_Intersect Transform Function does not require that you partition the data. However, you may improve performance by partitioning the data and using an OVER(PARTITION BEST) clause.

Spatial Indexes

The STV_Create_Index function can consume large amounts of processing time and memory. When you index new data for the first time, monitor memory usage to be sure it stays within

safe limits. Memory usage depends on:

- Number of polygons
- Number of vertices
- Amount of overlap among polygons

Recommendation	Details
Segment polygon data when a table contains a large number of polygons.	Segmenting the data allows the index creation process to run in parallel. This is advantageous because sometimes STV_Create_Index tasks cannot be completed when large tables that are not segmented prior to index creation.
Adjust STV_Create_Index parameters as needed for memory allocation and CPU usage.	<p>The <i>max_mem_mb</i> parameter can affect the resource usage of STV_Create_Index. <i>max_mem_mb</i> assigns a limit to the amount of memory that STV_Create_Index can allocate.</p> <p>Default value: 256</p> <p>Valid values: Any value less than or equal to the amount of memory in the GENERAL resource pool. Assigning a higher value results in an error.</p>
Make changes if STV_Create_Index cannot allocate 300 MB memory.	<p>Before STV_Create_Index starts creating the index, it tries to allocate about 300 MB of memory. If that much memory is not available, the function fails. If you get a failure message, try these solutions:</p> <ul style="list-style-type: none"> • Create the index at a time of less load on the system. • Avoid concurrent index creation. • Add more memory to your system.
Create misplaced indexes again, if needed.	When you back up your Vertica database, spatial index files are not included. If you misplace an index, use STV_Create_Index to re-create it.
Use STV_Refresh_Index to add new or updated polygons to an existing	Instead of rebuilding your spatial index each time you add new or updated polygons to a table, you

Recommendation	Details
index.	can use <code>STV_Refresh_Index</code> to append the polygons to your existing spatial index.

Checking Polygon Validity

Recommendation	Details
Run <code>ST_IsValid</code> to check if polygons are valid.	<p>Many spatial functions do not check the validity of polygons.</p> <ul style="list-style-type: none">• Run <code>ST_IsValid</code> on all polygons to determine if they are valid.• If your object is not valid, run <code>STV_IsValidReason</code> to get information about the location of the invalid polygon. <p>For more information, see Ensuring Polygon Validity Before Creating or Refreshing an Index.</p>

Spatial Objects

Vertica implements several data types for storing spatial objects, Well-Known Text (WKT) strings, and Well-Known Binary (WKB) representations. These data types include:

- [Supported Spatial Objects](#)
- [Spatial Reference Identifiers \(SRIDs\)](#)

Supported Spatial Objects

Vertica uses two spatial data types. These data types store two- and three-dimensional spatial objects in a table column:

- A GEOMETRY object is a spatial object with coordinates expressed as (x,y) pairs, defined in the Cartesian plane. All calculations use Cartesian coordinates.
- A GEOGRAPHY object is a spatial object defined as on the surface of a perfect sphere, or a spatial object in the WGS84 coordinate system. Coordinates are expressed in longitude/latitude angular values, measured in degrees. All calculations are in meters. For perfect sphere calculations, the sphere has a radius of 6371 kilometers, which approximates the shape of the earth.

Note: Other spatial programs may use an ellipsoid to model the earth, resulting in slightly different data.

The maximum size of a GEOMETRY or GEOGRAPHY data type is 10,000,000 bytes (10 MB). You cannot use a GEOMETRY or GEOGRAPHY type as the primary key for a table in a Vertica database.

Spatial Reference Identifiers (SRIDs)

A *spatial reference identifier* (SRID) is an integer value that represents a method for projecting coordinates on the plane. A SRID is metadata that indicates the coordinate system in which a spatial object is defined.

Geospatial functions using Geometry arguments must contain the same SRID. If the functions do not contain the same SRID, then the query returns an error.

For example, in this query the two points have different SRIDs. As a result the query returns an error:

```
=> SELECT ST_Distance(ST_GeomFromText('POINT(34 9)',2749), ST_GeomFromText('POINT(70 12)', 3359));  
ERROR 5861: Error calling processBlock() in User Function ST_Distance at [src/Distance.cpp:65],  
error code: 0, message: Geometries with different SRIDs found: 2749, 3359
```

Supported SRIDs

Vertica supports SRIDs derived from the EPSG standards. Geospatial functions using Geometry arguments must use supported SRIDs when performing calculations. SRID values of 0 to $2^{32}-1$ are valid. Queries with SRID values outside of this range will return an error.

Working with Spatial Objects in Tables

- [Defining Table Columns for Spatial Data](#)
- [Exporting Spatial Data from a Table](#)
- [Identifying Null Spatial Objects](#)
- [Loading Spatial Data from Shapefiles](#)
- [Loading Spatial Data into Tables Using COPY](#)
- [Retrieving Spatial Data from a Table as Well-Known Text \(WKT\)](#)

Defining Table Columns for Spatial Data

To define columns to contain GEOMETRY and GEOGRAPHY data, use this command:

```
=> CREATE TABLE [[db-name.]schema.]table-name (  
  column-name GEOMETRY[(length)],  
  column-name GEOGRAPHY[(length)]);
```

If you omit the length specification, the default column size is 1 MB. The maximum column size is 10 MB. The upper limit is not enforced, but the geospatial functions can only accept or return spatial data up to 10 MB.

You cannot modify the size or data type of a GEOMETRY or GEOGRAPHY column after creation. If the column size you created is not sufficient, create a new column with the desired size. Then copy the data from the old column, and drop the old column from the table.

You cannot import data to or export data from tables that contain spatial data from another Vertica database.

Important: A column width that is too large could impact performance. Use a column width that fits the data without being excessively large. See [STV_MemSize](#).

Exporting Spatial Data from a Table

You can export spatial data from a table in your Vertica database to a shapefile.

To export spatial data from a table to a shapefile:

1. As the superuser., set the shapefile export directory.

```
=> SELECT STV_SetExportShapefileDirectory(USING PARAMETERS path = '/home/geo/temp');
          STV_SetExportShapefileDirectory
-----
SUCCESS. Set shapefile export directory: [/home/geo/temp]
(1 row)
```

2. Export your spatial data to a shapefile.

```
=> SELECT STV_Export2Shapefile(*
          USING PARAMETERS shapefile = 'visualizations/city-data.shp',
                           shape = 'Polygon') OVER() FROM spatial_data;
Rows Exported | File Path
-----+-----
185873 | v_geo-db_node0001: /home/geo/temp/visualizations/city-data.shp
(1 row)
```

- The value asterisk (*) is the equivalent to listing all columns in the FROM clause.
- You can specify sub-directories when exporting your shapefile.
- Your shapefile must end with the file extension .shp.

3. Verify that three files now appear in the shapefile export directory.

```
$ ls
city-data.dbf  city-data.shp  city-data.shx
```

Identifying Null Spatial Objects

You can identify null GEOMETRY and GEOGRAPHY objects using the Vertica IS NULL and IS NOT NULL constructs.

This example uses the following table, where the row with `id=2` has a null value in the `geog` field.

```
=> SELECT id, ST_AsText(geom), ST_AsText(geog) FROM locations
       ORDER BY 1 ASC;
id | ST_AsText | ST_AsText
-----+-----+-----
1 | POINT (2 3) | POINT (-85 15)
2 | POINT (4 5) | 
3 | POLYGON ((-1 2, 0 3, 1 2, -1 2)) | POLYGON ((-24 12, -15 23, -20 27, -24 12))
4 | LINESTRING (-1 2, 1 5) | LINESTRING (-42.74 23.98, -62.19 23.78)
(4 rows)
```

Identify all the rows that have a null `geog` value:

```
=> SELECT id, ST_AsText(geom), (ST_AsText(geom) IS NULL) FROM locations
ORDER BY 1 ASC;
id |          ST_AsText          | ?column?
-----+-----+-----
 1 | POINT (2 3)                 | f
 2 | POINT (4 5)                 | t
 3 | POLYGON ((-1 2, 0 3, 1 2, -1 2)) | f
 4 | LINESTRING (-1 2, 1 5)      | f
(4 rows)
```

Identify the rows where the geom value is not null:

```
=> SELECT id, ST_AsText(geom), (ST_AsText(geom) IS NOT NULL) FROM locations
ORDER BY 1 ASC;
id |          st_astext          | ?column?
-----+-----+-----
 1 | POINT (2 3)                 | t
 2 | POINT (4 5)                 | f
 3 | LINESTRING (-1 2, 1 5)      | t
 4 | POLYGON ((-1 2, 0 3, 1 2, -1 2)) | t
(4 rows)
```

Loading Spatial Data from Shapefiles

Vertica provides the capability to load and parse spatial data that is stored in shapefiles. Shapefiles describe points, lines, and polygons. A shapefile is made up of three required files; all three files must be present and in the same directory to define the geometries:

- .shp—Contains the geometry data.
- .shx—Contains the positional index of the geometry.
- .dbf—Contains the attributes for each geometry.

To load spatial data from a shapefile:

1. Use `STV_ShpCreateTable` to generate a `CREATE TABLE` statement.

```
=> SELECT STV_ShpCreateTable ( USING PARAMETERS file = '/home/geo/temp/shp-files/spatial_
data.shp')
                                OVER() AS spatial_data;
                                spatial_data
-----+-----+-----
CREATE TABLE spatial_data(
  gid IDENTITY(64) PRIMARY KEY,
  uniq_id INT8,
  geom GEOMETRY(85)
);
(5 rows)
```

2. Create the table.

```
=> CREATE TABLE spatial_data(  
  gid IDENTITY(64) PRIMARY KEY,  
  uniq_id INT8,  
  geom GEOMETRY(85));
```

3. Load the shapefile.

```
=> COPY spatial_data WITH SOURCE STV_ShpSource(file='/home/geo/temp/shp-files/spatial_data.shp')  
  PARSER STV_ShpParser();  
Rows Loaded  
-----  
          10  
(1 row)
```

Supported Shapefile Shape Types

The following table lists the shapefile shape types that Vertica supports.

Shapefile Shape Type	Supported
Null shape	Yes
Point	Yes
Polyline	Yes
Polygon	Yes
MultiPoint	Yes
PointZ	No
PolylineZ	No
PolygonZ	No
MultiPointZ	No
PointM	No
PolylineM	No
PolygonM	No
MultiPointM	No

Shapefile Shape Type	Supported
MultiPatch	No

Loading Spatial Data into Tables Using COPY

You can load spatial data into a table in Vertica using a COPY statement.

To load data into Vertica using a COPY statement:

1. Create a table.

```
=> CREATE TABLE spatial_data (id INTEGER, geom GEOMETRY(200));  
CREATE TABLE
```

2. Create a text file named `spatial.dat` with the following data.

```
1|POINT(2 3)  
2|LINESTRING(-1 2, 1 5)  
3|POLYGON((-1 2, 0 3, 1 2, -1 2))
```

3. Use COPY to load the data into the table.

```
=> COPY spatial_data (id, gx FILLER LONG VARCHAR(605), geom AS ST_GeomFromText(gx)) FROM LOCAL  
'spatial.dat';  
Rows Loaded  
-----  
3  
(1 row)
```

The statement specifies a LONG VARCHAR(32000000) filler, which is the maximum size of WKT. You must specify a filler value large enough to hold the largest WKT you want to insert into the table.

Retrieving Spatial Data from a Table as Well-Known Text (WKT)

GEOMETRY and GEOGRAPHY data is stored in Vertica tables as LONG VARBINARY, which isn't human readable. You can use [ST_AsText](#) to return the spatial data as Well-Known Text (WKT).

To return spatial data as WKT:

```
=> SELECT id, ST_AsText(geom) AS WKT FROM spatial_data;  
id | WKT
```

```
-----+-----  
1 | POINT (2 3)  
2 | LINESTRING (-1 2, 1 5)  
3 | POLYGON ((-1 2, 0 3, 1 2, -1 2))  
(3 rows)
```

Working with GeoHash Data

Vertica supports [GeoHashes](#). A GeoHash is a geocoding system for hierarchically encoding increasingly granular spatial references. Each additional character in a GeoHash drills down to a smaller section of a map.

You can use Vertica to generate spatial data from GeoHashes and GeoHashes from spatial data. Vertica supports the following functions for use with GeoHashes:

- [ST_GeoHash](#) - Returns a GeoHash in the shape of the specified geometry.
- [ST_GeomFromGeoHash](#) - Returns a polygon in the shape of the specified GeoHash.
- [ST_PointFromGeoHash](#) - Returns the center point of the specified GeoHash.

For example, to generate a full precision and partial precision GeoHash from a single point.

```
=> SELECT ST_GeoHash(ST_GeographyFromText('POINT(3.14 -1.34)'), LENGTH(ST_GeoHash(ST_
GeographyFromText('POINT(3.14 -1.34)'))),
           ST_GeoHash(ST_GeographyFromText('POINT(3.14 -1.34)') USING PARAMETERS
numchars=5) partial_hash;
   ST_GeoHash | LENGTH | partial_hash
-----+-----+-----
 kpf0rk3zmcswks75010 |    20 | kpf0r
(1 row)
```

This example shows how to generate a GeoHash from a multipoint point object. The returned polygon is a geometry object of the smallest tile that encloses that GeoHash.

```
=> SELECT ST_AsText(ST_GeomFromGeoHash(ST_GeoHash(ST_GeomFromText('MULTIPOINT(0 0, 0.0002
0.0001)')))) AS region_1,
           ST_AsText(ST_GeomFromGeoHash(ST_GeoHash(ST_GeomFromText('MULTIPOINT(0.0001
0.0001, 0.0003 0.0002)')))) AS region_2;
-[ RECORD 1 ]-----+-----+-----
   region_1 | POLYGON ((0 0, 0.00137329101562 0, 0.00137329101562 0.00137329101562, 0
0.00137329101562, 0 0))
   region_2 | POLYGON ((0 0, 0.010986328125 0, 0.010986328125 0.0054931640625, 0 0.0054931640625, 0
0))
```

Spatial Joins with ST_Intersects and STV_Intersect

Spatial joins allow you to identify spatial relationships between two sets of spatial data. For example, you can use spatial joins to:

- Calculate the density of mobile calls in various regions to determine the location of a new cell phone tower.
- Identify homes that fall within the impact zone of a hurricane.
- Calculate the number of users who live within a certain ZIP code.
- Calculate the number of customers in a retail store at any given time.

Best Practices for Spatial Joins

Use these best practices to improve overall performance and optimize your spatial queries.

Best practices for using spatial joins in Vertica include:

- Table segmentation to speed up index creation
- Adequately sizing a geometry column to store point data
- Loading Well-Known Text (WKT) directly into a Geometry column, using STV_GeometryPoint in a COPY statement
- Using OVER (PARTITION BEST) with STV_Intersect transform queries

Best Practices Example

Note: The following example was originally published in an Vertica blog post about using spatial data in museums. To read the entire blog, go to:

<https://my.vertica.com/blog/using-location-data-with-hp-vertica-placeba-p227636/>

Before performing the steps in the following example, download `place_output.csv.zip` from the Vertica Place GitHub repository (<https://github.com/vertica/Vertica-Geospatial>). You need to use the data set from this repository.

1. Create the table for the polygons. Use a GEOMETRY column width that fits your data without being excessively large. A good column-width fit improves performance. In addition, segmenting the table by HASH provides the advantages of parallel computation.

```
=> CREATE TABLE artworks (gid int, g GEOMETRY(700)) SEGMENTED BY HASH(gid) ALL NODES;
```

2. Use a copy statement with ST_Buffer to create and load the polygons on which to run the intersect. By using ST_Buffer in your copy statement, you can use that function to create the polygons.

```
=> COPY artworks(gid, gx FILLER LONG VARCHAR, g AS ST_Buffer(ST_GeomFromText(gx),8)) FROM STDIN  
DELIMITER ',';  
>> 1, POINT(10 45)  
>> 2, POINT(25 45)  
>> 3, POINT(35 45)  
>> 4, POINT(35 15)  
>> 5, POINT(30 5)  
>> 6, POINT(15 5)  
>> \.
```

3. Create a table for the location data, represented by points. You can store point data in a GEOMETRY column of 100 bytes. Avoid over-fitting your GEOMETRY column. Doing so can significantly degrade spatial intersection performance. Also, segment this table by HASH, to take advantage of parallel computation.

```
=> CREATE TABLE usr_data (gid identity, usr_id int, date_time timestamp, g GEOMETRY(100))  
SEGMENTED BY HASH(gid) ALL NODES;
```

4. During the copy statement, transform the raw location data to GEOMETRY data. You must perform this transformation because your location data needs to use the GEOMETRY data type. Use the function STV_GeometryPoint to transform the x and y columns of the source table.

```
=> COPY usr_data (usr_id, date_time, x FILLER LONG VARCHAR,  
y FILLER LONG VARCHAR, g AS STV_GeometryPoint(x, y))  
FROM LOCAL 'place_output.csv' DELIMITER ',' ENCLOSED BY '';
```

5. Create the spatial index for the polygons. This index helps you speed up intersection calculations.

```
=> SELECT STV_Create_Index(gid, g USING PARAMETERS index='art_index', overwrite=true) OVER() FROM  
artworks;
```

6. Write an analytic query that returns the number of intersections per polygon. Specify that Vertica ignore any usr_id that intersects less than 20 times with a given polygon.

```
=> SELECT pol_gid,  
        COUNT(DISTINCT(usr_id)) AS count_user_visit  
FROM  
  (SELECT pol_gid,  
         usr_id,  
         COUNT(usr_id) AS user_points_in  
   FROM  
     (SELECT STV_Intersect(usr_id, g USING PARAMETERS INDEX='art_index') OVER(PARTITION BEST) AS  
      (usr_id,  
       pol_gid)  
    FROM usr_data  
     WHERE date_time BETWEEN '2014-07-02 09:30:20' AND '2014-07-02 17:05:00') AS c  
   GROUP BY pol_gid,  
            usr_id HAVING COUNT(usr_id) > 20) AS real_visits  
GROUP BY pol_gid  
ORDER BY count_user_visit DESC;
```

Optimizations in the Example Query

This query has the following optimizations:

- The time predicated appears in the subquery.
- Using the location data table avoids the need for an expensive join.
- The query uses OVER (PARTITION BEST), to improve performance by partitioning the data.
- The user_points_in provides an estimate of the combined time spent intersecting with the artwork by all visitors.

Ensuring Polygon Validity Before Creating or Refreshing an Index

When Vertica creates or updates a spatial index it does not check polygon validity. To prevent getting invalid results when you query your spatial index, you should check the validity of your polygons prior to creating or updating your spatial index.

The following example shows you how to check the validity of polygons.

1. Create a table and load spatial data.

```
=> CREATE TABLE polygon_validity_test (gid INT, geom GEOMETRY);  
CREATE TABLE  
=> COPY polygon_validity_test (gid, gx FILLER LONG VARCHAR, geom AS St_GeomFromText(gx)) FROM  
STDIN;
```

```
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 2|POLYGON((-31 74,8 70,8 50,-36 53,-31 74))  
>> 3|POLYGON((-38 50,4 13,11 45,0 65,-38 50))  
>> 4|POLYGON((-12 42,-12 42,27 48,14 26,-12 42))  
>> 5|POLYGON((0 0,1 1,0 0,2 1,1 1,0 0))  
>> 6|POLYGON((3 3,2 2,2 1,2 3,3 3))  
>> \.
```

2. Use ST_IsValid and STV_IsValidReason to find any invalid polygons.

```
=> SELECT gid, ST_IsValid(geom), STV_IsValidReason(geom) FROM polygon_validity_test;  
gid | ST_IsValid | STV_IsValidReason  
-----+-----+-----  
4 | t |  
6 | f | Self-intersection at or near POINT (2 1)  
2 | t |  
3 | t |  
5 | f | Self-intersection at or near POINT (0 0)  
(5 rows)
```

Now that we have identified the invalid polygons in our table, there are a couple different ways you can handle the invalid polygons when creating or refreshing a spatial index.

Filtering Invalid Polygons Using a WHERE Clause

This method is slower than filtering before creating an index because it checks the validity of each polygon at execution time.

The following example shows you how to exclude invalid polygons using a WHERE clause.

```
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index = 'valid_polygons') OVER()  
FROM polygon_validity_test  
WHERE ST_IsValid(geom) = 't';
```

Filtering Invalid Polygons Before Creating or Refreshing an Index

This method is faster than filtering using a WHERE clause because you incur the performance cost prior to building the index.

The following example shows you how to exclude invalid polygons by creating a new table excluding invalid polygons.

```
=> CREATE TABLE polygon_validity_clean AS  
SELECT *  
FROM polygon_validity_test  
WHERE ST_IsValid(geom) = 't';
```

```
CREATE TABLE
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index = 'valid_polygons') OVER()
FROM polygon_validity_clean;
```

STV_Intersect: Scalar Function vs. Transform Function

The STV_Intersect functions are similar in purpose, but you use them differently.

STV_Intersect Function Type	Description	Performance
Scalar	Matches a point to a polygon. If several polygons contain the point, this function returns a <code>gid</code> value. The result is a polygon <code>gid</code> or, if no polygon contains the point, the result is NULL.	Eliminates points that do not intersect with any indexed polygons, avoiding unnecessary comparisons.
Transform	Matches a point to all the polygons that contain it. When a point does not intersect with any polygon in the index, the function returns no rows.	Processes all input points regardless of whether or not they intersect with the indexed polygons.

In the following example, the STV_Intersect scalar function compares the points in the `points` table to the polygons in a spatial index named `my_polygons`. STV_Intersect returns all points and polygons that match exactly:

```
=> SELECT gid AS pt_gid
      STV_Intersect(geom USING PARAMETERS index='my_polygons') AS pol_gid
FROM points ORDER BY pt_gid;
pt_gid | pol_gid
-----+-----
  100 |      2
  101 |
  102 |      2
  103 |
  104 |
  105 |      3
  106 |
  107 |
(8 rows)
```

The following example shows how to use the STV_Intersect transform function to return information about the three point-polygon pairs that match and each of the polygons they match:

```
=> SELECT STV_Intersect(gid, geom
  USING PARAMETERS index='my_polygons')
  OVER (PARTITION BEST) AS (pt_gid, pol_id)
  FROM points;
pt_gid | pol_id
-----+-----
  100 |     1
  100 |     2
  100 |     3
  102 |     2
  105 |     3
(3 rows)
```

See Also

- [STV_Intersect Scalar Function](#)
- [STV_Intersect Transform Function](#)

Performing Spatial Joins with STV_Intersect Functions

Suppose you want to process a medium-to-large spatial data set and determine which points intersect with which polygons. In that case, first create a spatial index using `STV_Create_Index`. A spatial index provides efficient access to the set of polygons.

Then, use the `STV_Intersect` scalar or transform function to identify which point-polygon pairs match.

Spatial Indexes and STV_Intersect

Before performing a spatial join using one of the `STV_Intersect` functions, you must first run `STV_Create_Index` to create a database object that contains information about polygons. This object is called a *spatial index* of the set of polygons. The spatial index improves the time it takes for the `STV_Intersect` functions to access the polygon data.

Vertica creates spatial indexes in a global space. Thus, any user with access to the `STV_*_Index` functions can describe, rename, or drop indexes created by any other user.

Vertica provides functions that work with spatial indexes:

- [STV_Create_Index](#)—Stores information about polygons in an index to improve performance.
- [STV_Describe_Index](#)—Retrieves information about an index.
- [STV_Drop_Index](#)—Deletes a spatial index.

- [STV_Refresh_Index](#)—Refreshes a spatial index.
- [STV_Rename_Index](#)—Renames a spatial index.

When to Use ST_Intersects vs. STV_Intersect

Vertica provides two capabilities to identify whether a set of points intersect with a set of polygons. Depending on the size of your data set, choose the approach that gives the best performance:

- When comparing a set of geometries to a single geometry to see if they intersect, use the [ST_Intersects](#) function.
- To determine if a set of points intersects with a set of polygons in a medium-to-large data set, first create a spatial index using [STV_Create_Index](#). Then, use one of the [STV_Intersect](#) functions to return the set of pairs that intersect.

Note: You can only perform spatial joins on GEOMETRY data.

Performing Spatial Joins with ST_Intersects

The [ST_Intersects](#) function determines if two GEOMETRY objects intersect or touch at a single point.

Use [ST_Intersects](#) when you want to identify if a small set of geometries in a column intersect with a given geometry.

Example

The following example uses `ST_Intersects` to compare a column of point geometries to a single polygon. The table that contains the points has 1 million rows.

`ST_Intersects` returns only the points that intersect with the polygon. Those points represent about 0.01% of the points in the table:

```
=> CREATE TABLE points_1m(gid IDENTITY, g GEOMETRY(100)) ORDER BY gid;
=> COPY points_1m(wkt FILLER LONG VARCHAR(100), g AS ST_GeomFromText(wkt))
  FROM LOCAL '/data/points.dat' DIRECT;
Rows Loaded
-----
      1000000
(1 row)
=> SELECT ST_AsText(g) FROM points_1m WHERE
  ST_Intersects
  (
    g,
    ST_GeomFromText('POLYGON((-71 42, -70.9 42, -70.9 42.1, -71 42.1, -71 42))')
  );
      st_astext
-----
POINT (-70.97532 42.03538)
POINT (-70.97421 42.0376)
POINT (-70.99004 42.07538)
POINT (-70.99477 42.08454)
POINT (-70.99088 42.08177)
POINT (-70.98643 42.07593)
POINT (-70.98032 42.07982)
POINT (-70.95921 42.00982)
POINT (-70.95115 42.02177)
...
(116 rows)
```

Micro Focus recommends that you test the intersections of two columns of geometries by creating a spatial index. Use one of the `STV_Intersect` functions as described in [STV_Intersect: Scalar Function vs. Transform Function](#).

Working with Spatial Objects from Client Applications

The Vertica client driver libraries provide interfaces for connecting your client applications to your Vertica database. The drivers simplify exchanging data for loading, report generation, and other common database tasks.

There are three separate client drivers:

- Open Database Connectivity (ODBC)—The most commonly-used interface for third-party applications and clients written in C, Python, PHP, Perl, and most other languages.
- Java Database Connectivity (JDBC)—Used by clients written in the Java programming language.
- ActiveX Data Objects for .NET (ADO.NET)—Used by clients developed using Microsoft's .NET Framework and written in C#, Visual Basic .NET, and other .NET languages.

Vertica Place supports the following new data types:

- LONG VARCHAR
- LONG VARBINARY
- GEOMETRY
- GEOGRAPHY

The client driver libraries support these data types; the following sections describe that support and provide examples.

Using LONG VARCHAR and LONG VARBINARY Data Types with ODBC

The ODBC drivers support the LONG VARCHAR and LONG VARBINARY data types similarly to VARCHAR and VARBINARY data types. When binding input or output parameters to a LONG VARCHAR or LONG VARBINARY column in a query, use the SQL_LONGVARCHAR and SQL_LONGVARBINARY constants to set the column's data type. For example, to bind an input parameter to a LONG VARCHAR column, you would use a statement that looks like this:

```
rc = SQLBindParameter(hdIStmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR,
    80000, 0, (SQLPOINTER)myLongString, sizeof(myLongString), NULL);
```

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

Using LONG VARCHAR and LONG VARBINARY Data Types with JDBC

Using LONG VARCHAR and LONG VARBINARY data types in a JDBC client application is similar to using VARCHAR and VARBINARY data types. The JDBC driver transparently handles the conversion (for example, between a Java `String` object and a LONG VARCHAR). The following example code demonstrates inserting and retrieving a LONG VARCHAR string. It uses the JDBC `Types` class to determine the data type of the string returned by Vertica, although it does not actually need to know whether the database column is a LONG VARCHAR or just a VARCHAR in order to retrieve the value.

```
import java.sql.*;
import java.util.Properties;

public class LongVarcharExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.vertica.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Could not find the JDBC driver class.");
            e.printStackTrace();
            return;
        }
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",
                myProp);
            // establish connection and make a table for the data.
            Statement stmt = conn.createStatement();

            // How long we want the example string to be. This is
            // larger than can fit into a traditional VARCHAR (which is limited
            // to 65000.
            int length = 100000;

            // Create a table with a LONG VARCHAR column that can store
            // the string we want to insert.
            stmt.execute("DROP TABLE IF EXISTS longtable CASCADE");
            stmt.execute("CREATE TABLE longtable (text LONG VARCHAR(" + length
                + "))");
            // Build a long string by appending an integer to a string builder
            // until we hit the size limit. Will result in a string
            // containing 01234567890123....
            StringBuilder sb = new StringBuilder(length);
            for (int i = 0; i < length; i++)
            {
                sb.append(i % 10);
            }
        }
    }
}
```

```
}
String value = sb.toString();

System.out.println("String value is " + value.length() +
    " characters long.");

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO longtable (text)" +
    " VALUES(?)");
try {
    // Insert LONG VARCHAR value
    System.out.println("Inserting LONG VARCHAR value");
    pstmt.setString(1, value);
    pstmt.addBatch();
    pstmt.executeBatch();

    // Query the table we created to get the value back.
    ResultSet rs = null;
    rs = stmt.executeQuery("SELECT * FROM longtable");

    // Get metadata about the result set.
    ResultSetMetaData rsmd = rs.getMetaData();
    // Print the type of the first column. Should be
    // LONG VARCHAR. Also check it against the Types class, to
    // recognize it programmatically.
    System.out.println("Column #1 data type is: " +
        rsmd.getColumnTypeName(1));
    if (rsmd.getColumnType(1) == Types.LONGVARCHAR) {
        System.out.println("It is a LONG VARCHAR");
    } else {
        System.out.println("It is NOT a LONG VARCHAR");
    }
}

// Print out the string length of the returned value.
while (rs.next()) {
    // Use the same getString method to get the value that you
    // use to get the value of a VARCHAR.
    System.out.println("Returned string length: " +
        rs.getString(1).length());
}
} catch (SQLException e) {
    System.out.println("Error message: " + e.getMessage());
    return; // Exit if there was an error
}
// Cleanup
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you

attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

Using GEOMETRY and GEOGRAPHY Data Types in ODBC

Vertica GEOMETRY and GEOGRAPHY data types are backed by LONG VARBINARY native types and ODBC client applications treat them as binary data. However, these data types have a format that is unique to Vertica. To manipulate this data in your C++ application, you must use the functions in Vertica that convert them to a recognized format.

To convert a WKT or WKB to the GEOMETRY or GEOGRAPHY format, use one of the following SQL functions:

- [ST_GeographyFromText](#)—Converts a WKT to a GEOGRAPHY type.
- [ST_GeographyFromWKB](#)—Converts a WKB to a GEOGRAPHY type.
- [ST_GeomFromText](#)—Converts a WKT to a GEOMETRY type.
- [ST_GeomFromWKB](#)—Converts a WKB to GEOMETRY type.

To convert a GEOMETRY or GEOGRAPHY object to its corresponding WKT or WKB, use one of the following SQL functions:

- [ST_AsText](#)—Converts a GEOMETRY or GEOGRAPHY object to a WKT, returns a LONGVARCHAR.
- [ST_AsBinary](#)—Converts a GEOMETRY or GEOGRAPHY object to a WKB, returns a LONG VARBINARY.

The following code example converts WKT data into GEOMETRY data using `ST_GeomFromText` and stores it in a table. Later, this example retrieves the GEOMETRY data from the table and converts it to WKT and WKB format using `ST_AsText` and `ST_AsBinary`.

```
// Compile on Linux using:  
// g++ -g -I/opt/vertica/include -L/opt/vertica/lib64 -lodbc -o SpatialData SpatialData.cpp  
// Some standard headers  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <assert.h>  
#include <sstream>  
// Only needed for Windows clients
```

```
// #include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
// Helper function to print SQL error messages.
template <typename HandleT>
void reportError(int handleTypeEnum, HandleT hdl)
{
    // Get the status records.
    SQLSMALLINT    i, MsgLen;
    SQLRETURN      ret2;
    SQLCHAR        SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER     NativeError;
    i = 1;
    printf("\n");
    while ((ret2 = SQLGetDiagRec(handleTypeEnum, hdl, i, SqlState, &NativeError,
                               Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA) {
        printf("error record %d\n", i);
        printf("sqlstate: %s\n", SqlState);
        printf("detailed msg: %s\n", Msg);
        printf("native error code: %d\n\n", NativeError);
        i++;
    }
    exit(EXIT_FAILURE); // bad form... but Ok for this demo
}
int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    assert(SQL_SUCCEEDED(ret));
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
                       (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    assert(SQL_SUCCEEDED(ret));
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    assert(SQL_SUCCEEDED(ret));
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char* userID = "dbadmin";
    const char* passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
                    SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
                    (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not connect to database.\n");
        reportError<SQLHDBC>(SQL_HANDLE_DBC, hdlDbc);
    } else {
        printf("Connected to database.\n");
    }
}

// Disable AUTOCOMMIT
ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF,
                       SQL_NTS);
```

```
// Set up a statement handle
SQLHSTMT hd1stmt;
SQLAllocHandle(SQL_HANDLE_STMT, hd1dbc, &hd1stmt);

// Drop any previously defined table.
ret = SQLExecDirect(hd1stmt, (SQLCHAR*)"DROP TABLE IF EXISTS polygons",
    SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1stmt);}

// Run query to create a table to hold a geometry.
ret = SQLExecDirect(hd1stmt,
    (SQLCHAR*)"CREATE TABLE polygons(id INTEGER PRIMARY KEY, poly GEOMETRY);",
    SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1stmt);}

// Create the prepared statement. This will insert data into the
// table we created above. It uses the ST_GeomFromText function to convert the
// string-formatted polygon definition to a GEOMETRY datatype.
printf("Creating prepared statement\n");
ret = SQLPrepare (hd1stmt,
    (SQLCHAR*)"INSERT INTO polygons(id, poly) VALUES(?, ST_GeomFromText(?))",
    SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1stmt);}

SQLINTEGER id = 0;
int numBatches = 5;
int rowsPerBatch = 10;

// Polygon definition as a string.
char polygon[] = "polygon((1 1, 1 2, 2 2, 2 1, 1 1))";
// Bind variables to the parameters in the prepared SQL statement
ret = SQLBindParameter(hd1stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
    0, 0, &id, 0, NULL);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT,hd1stmt);}
// Bind polygon string to the geometry column
SQLBindParameter(hd1stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR,
    strlen(polygon), 0, (SQLPOINTER)polygon, strlen(polygon), NULL);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT,hd1stmt);}
// Execute the insert
ret = SQLExecute(hd1stmt);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1stmt);
} else {
    printf("Executed batch.\n");
}

// Commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hd1dbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1stmt);
} else {
    printf("Committed transaction\n");
}

// Now, create a query to retrieve the geometry.
ret = SQLAllocHandle(SQL_HANDLE_STMT, hd1dbc, &hd1stmt);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1stmt);}
printf("Getting data from table.\n");
// Execute a query to get the id, raw geometry data, and
```

```
// the geometry data as a string. Uses the ST_AsText SQL function to
// format raw data back into a string polygon definition
ret = SQLExecDirect(hdlStmt,
    (SQLCHAR*)"select id,ST_AsBinary(poly),ST_AsText(poly) from polygons ORDER BY id;",
    SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}

SQLINTEGER idval;
// 10MB buffer to hold the raw data from the geometry (10Mb is the maximum
// length of a GEOMETRY)
SQLCHAR* polygonval = (SQLCHAR*)malloc(10485760);
SQLLEN polygonlen, polygonstrlen;
// Buffer to hold a LONGVARCHAR that can result from converting the
// geometry to a string.
SQLTCHAR* polygonstr = (SQLTCHAR*)malloc(33554432);

// Get the results of the query and print each row.
do {
    ret = SQLFetch(hdlStmt);
    if (SQL_SUCCEEDED(ret)) {
        // ID column
        ret = SQLGetData(hdlStmt, 1, SQL_C_LONG, &idval, 0, NULL);
        if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
        printf("id: %d\n", idval);
        // The WKB format geometry data
        ret = SQLGetData(hdlStmt, 2, SQL_C_BINARY, polygonval, 10485760,
            &polygonlen);
        if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
        printf("Polygon in WKB format: ");
        // Print each byte of polygonval buffer in hex format.
        for (int z = 0; z < polygonlen; z++)
            printf("%02x ", polygonval[z]);
        printf("\n");
        // Geometry data formatted as a string.
        ret = SQLGetData(hdlStmt, 3, SQL_C_TCHAR, polygonstr, 33554432, &polygonstrlen);
        if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
        printf("Polygon in WKT format: %s\n", polygonstr);
    }
} while(SQL_SUCCEEDED(ret));

free(polygonval);
free(polygonstr);
// Clean up
printf("Free handles.\n");
ret = SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
ret = SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
exit(EXIT_SUCCESS);
}
```

The output of running the above example is:

```
Connecting to database.
Connected to database.
```


The following code example converts WKT and WKB data into GEOMETRY data using `ST_GeomFromText` and `ST_GeomFromWKB` and stores it in a table. Later, this example retrieves the GEOMETRY data from the table and converts it to WKT and WKB format using `ST_AsText` and `ST_AsBinary`.

```
import java.io.InputStream;
import java.io.Reader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
public class GeospatialDemo
{
    public static void main(String [] args) throws Exception
    {
        Class.forName("com.vertica.jdbc.Driver");
        Connection conn =
            DriverManager.getConnection("jdbc:vertica://localhost:5433/db",
                                      "user", "password");

        conn.setAutoCommit(false);

        Statement stmt = conn.createStatement();
        stmt.execute("CREATE TABLE polygons(id INTEGER PRIMARY KEY, poly GEOMETRY)");

        int id = 0;
        int numBatches = 5;
        int rowsPerBatch = 10;

        //batch inserting WKT data
        PreparedStatement pstmt = conn.prepareStatement("INSERT INTO polygons
            (id, poly) VALUES(?, ST_GeomFromText(?))");
        for(int i = 0; i < numBatches; i++)
        {
            for(int j = 0; j < rowsPerBatch; j++)
            {
                //Insert your own WKT data here
                pstmt.setInt(1, id++);
                pstmt.setString(2, "polygon((1 1, 1 2, 2 2, 2 1, 1 1))");
                pstmt.addBatch();
            }
            pstmt.executeBatch();
        }

        conn.commit();
        pstmt.close();
        //batch insert WKB data
        pstmt = conn.prepareStatement("INSERT INTO polygons(id, poly)
            VALUES(?, ST_GeomFromWKB(?))");
        for(int i = 0; i < numBatches; i++)
        {
            for(int j = 0; j < rowsPerBatch; j++)
            {
                //Insert your own WKB data here
                byte [] wkb = getWKB();
                pstmt.setInt(1, id++);
                pstmt.setBytes(2, wkb);
            }
        }
    }
}
```

```
        pstmt.addBatch();
    }
    pstmt.executeBatch();
}

conn.commit();
pstmt.close();
//selecting data as WKT
ResultSet rs = stmt.executeQuery("select ST_AsText(poly) from polygons");
while(rs.next())
{
    String wkt = rs.getString(1);
    Reader wktReader = rs.getCharacterStream(1);
    //process the wkt as necessary
}
rs.close();

//selecting data as WKB
rs = stmt.executeQuery("select ST_AsBinary(poly) from polygons");
while(rs.next())
{
    byte [] wkb = rs.getBytes(1);
    InputStream wkbStream = rs.getBinaryStream(1);
    //process the wkb as necessary
}
rs.close();

//binding parameters in predicates
pstmt = conn.prepareStatement("SELECT id FROM polygons WHERE
                             ST_Contains(ST_GeomFromText(?), poly)");
pstmt.setString(1, "polygon((1 1, 1 2, 2 2, 2 1, 1 1))");
rs = pstmt.executeQuery();
while(rs.next())
{
    int pk = rs.getInt(1);
    //process the results as necessary
}
rs.close();

conn.close();
}
}
```

Using GEOMETRY and GEOGRAPHY Data Types in ADO.NET

Vertica GEOMETRY and GEOGRAPHY data types are backed by LONG VARBINARY native types and ADO.NET client applications treat them as binary data. However, these data types have a format that is unique to Vertica. To manipulate this data in your C# application, you must use the functions in Vertica that convert them to a recognized format.

To convert a WKT or WKB to the GEOMETRY or GEOGRAPHY format, use one of the following SQL functions:

- [ST_GeographyFromText](#)—Converts a WKT to a GEOGRAPHY type.
- [ST_GeographyFromWKB](#)—Converts a WKB to a GEOGRAPHY type.
- [ST_GeomFromText](#)—Converts a WKT to a GEOMETRY type.
- [ST_GeomFromWKB](#)—Converts a WKB to GEOMETRY type.

To convert a GEOMETRY or GEOGRAPHY object to its corresponding WKT or WKB, use one of the following SQL functions:

- [ST_AsText](#)—Converts a GEOMETRY or GEOGRAPHY object to a WKT, returns a LONGVARCHAR.
- [ST_AsBinary](#)—Converts a GEOMETRY or GEOGRAPHY object to a WKB, returns a LONG VARBINARY.

The following C# code example converts WKT data into GEOMETRY data using `ST_GeomFromText` and stores it in a table. Later, this example retrieves the GEOMETRY data from the table and converts it to WKT and WKB format using `ST_AsText` and `ST_AsBinary`.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder =
                new VerticaConnectionStringBuilder();
            builder.Host = "VerticaHost";
            builder.Database = "VMart";
            builder.User = "ExampleUser";
            builder.Password = "password123";
            VerticaConnection _conn = new
                VerticaConnection(builder.ToString());
            _conn.Open();

            VerticaCommand command = _conn.CreateCommand();
            command.CommandText = "DROP TABLE IF EXISTS polygons";
            command.ExecuteNonQuery();
            command.CommandText =
                "CREATE TABLE polygons (id INTEGER PRIMARY KEY, poly GEOMETRY)";
            command.ExecuteNonQuery();
            // Prepare to insert a polygon using a prepared statement. Use the
            // ST_GeomFromText SQL function to convert from WKT to GEOMETRY.
            VerticaTransaction txn = _conn.BeginTransaction();
```

```
command.CommandText =
    "INSERT into polygons VALUES(@id, ST_GeomFromText(@polygon));";
command.Parameters.Add(new
    VerticaParameter("id", VerticaType.BigInt));
command.Parameters.Add(new
    VerticaParameter("polygon", VerticaType.VarChar));
command.Prepare();
// Set the values for the parameters
command.Parameters["id"].Value = 0;
//
command.Parameters["polygon"].Value =
    "polygon((1 1, 1 2, 2 2, 2 1, 1 1));";
// Execute the query to insert the value
command.ExecuteNonQuery();

// Now query the table
VerticaCommand query = _conn.CreateCommand();
query.CommandText =
    "SELECT id, ST_AsText(poly), ST_AsBinary(poly) FROM polygons;";
VerticaDataReader dr = query.ExecuteReader();
while (dr.Read())
{
    Console.WriteLine("ID: " + dr[0]);
    Console.WriteLine("Polygon WKT format data type: "
        + dr.GetDataTypeName(1) +
        " Value: " + dr[1]);
    // Get the WKB format of the polygon and print it out as hex.
    Console.WriteLine("Polygon WKB format data type: "
        + dr.GetDataTypeName(2));
    Console.WriteLine(" Value: "
        + BitConverter.ToString((byte[])dr[2]));
}
_conn.Close();
}
}
```

The example code prints the following on the system console:

```
ID: 0
Polygon WKT format data type: LONG VARCHAR Value: POLYGON ((1 1, 1 2,
2 2, 2 1,1 1))
Polygon WKB format data type: LONG VARBINARY Value: 01-03-00-00-00-01
-00-00-00-05-00-00-00-00-00-00-00-00-00-00-F0-3F-00-00-00-00-00-00-F0-3F
-00-00-00-00-00-00-F0-3F-00-00-00-00-00-00-00-00-40-00-00-00-00-00-00
-40-00-00-00-00-00-00-00-40-00-00-00-00-00-00-00-40-00-00-00-00-00
-F0-3F-00-00-00-00-00-00-F0-3F-00-00-00-00-00-00-F0-3F
```

OGC Spatial Definitions

Using Vertica requires an understanding of the Open Geospatial Consortium (OGC) concepts and capabilities. For more information, see the [OGC Simple Feature Access Part 1 - Common Architecture](#) specification.

Spatial Classes

Vertica supports several classes of objects, as defined in the OGC standards.

Point

A location in two-dimensional space that is identified by one of the following:

- X and Y coordinates
- Longitude and latitude values

A point has dimension 0 and no boundary.

Examples

The following example uses a GEOMETRY point:

```
=> CREATE TABLE point_geo (gid int, geom GEOMETRY(100));
CREATE TABLE
=> COPY point_geo(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1, POINT(3 5)
>>\.
=> SELECT gid, ST_AsText(geom) FROM point_geo;
gid | ST_AsText
-----+-----
  1 | POINT (3 5)
(1 row)
```

The following example uses a GEOGRAPHY point:

```
=> CREATE TABLE point_geog (gid int, geog geography(100));
CREATE TABLE
=> COPY point_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin
delimiter ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1, POINT(42 71)
>>\.
=> SELECT gid, ST_AsText(geog) FROM point_geog;
gid | ST_AsText
-----+-----
  1 | POINT (42 71)
(1 row)
```

Multipoint

A set of one or more points. A multipoint object has dimension 0 and no boundary.

Examples

The following example uses a GEOMETRY multipoint:

```
=> CREATE TABLE mpoint_geo (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY mpoint_geo(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter
'|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|MULTIPOINT(4 7, 8 10)
>>\.
=> SELECT gid, ST_AsText(geom) FROM mpoint_geo;
gid |          st_astext
-----+-----
  1 | MULTIPOINT (7 8, 6 9)
(1 row)
```

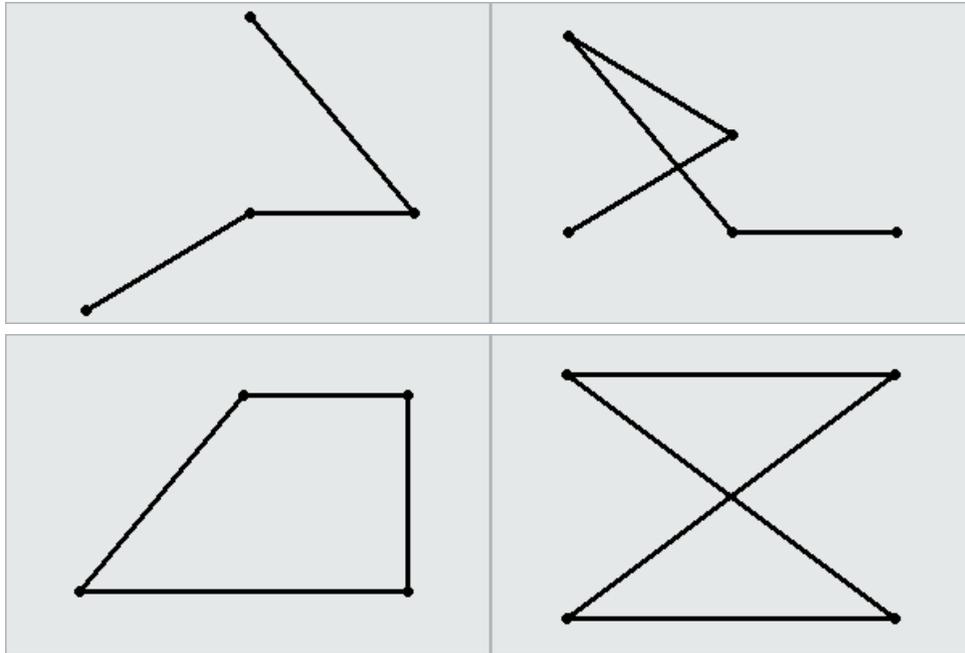
The following example uses a GEOGRAPHY multipoint:

```
=> CREATE TABLE mpoint_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY mpoint_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|MULTIPOINT(42 71, 41.4 70)
>>\.
=> SELECT gid, ST_AsText(geom) FROM mpoint_geo;
gid |          st_astext
-----+-----
  1 | MULTIPOINT (42 71, 41.4 70)
(1 row)
```

Linestring

One or more connected lines, identified by pairs of consecutive points. A linestring has dimension 1. The boundary of a linestring is a multipoint object containing its start and end points.

The following are examples of linestrings:



Examples

The following example uses the GEOMETRY type to create a table, use copy to load a linestring to the table, and then queries the table to view the linestring:

```
=> CREATE TABLE linestring_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY linestring_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|LINESTRING(0 0, 1 1, 2 2, 3 4, 2 4, 1 5)
>>\.
=> SELECT gid, ST_AsText(geom) FROM linestring_geom;
gid |          ST_AsText
-----+-----
  1 | LINESTRING (0 0, 1 1, 2 2, 3 4, 2 4, 1 5)
(1 row)
```

The following example uses the GEOGRAPHY type to create a table, use copy to load a linestring to the table, and then queries the table to view the linestring:

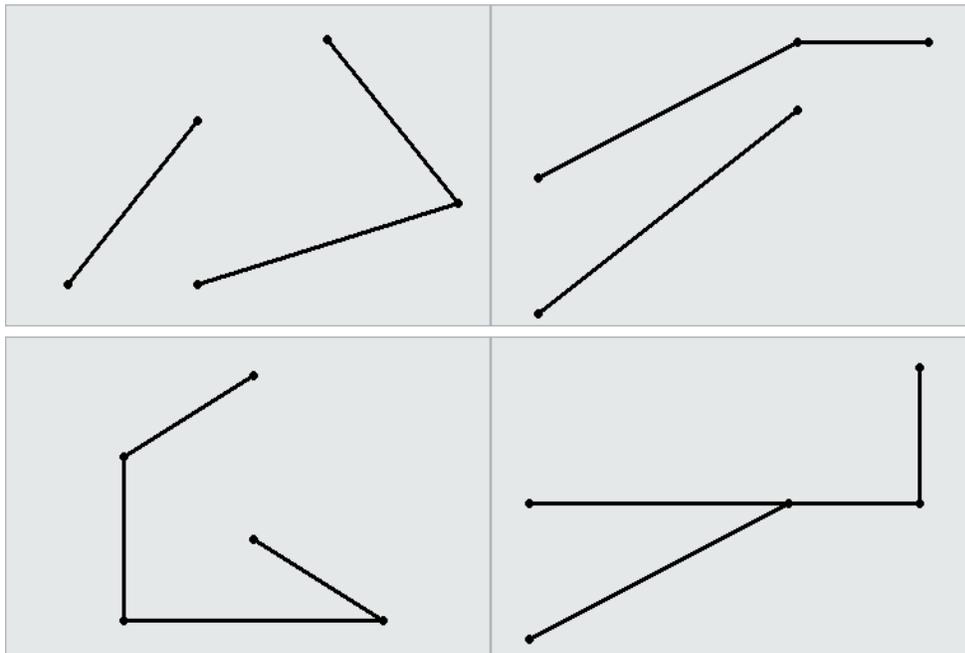
```
=> CREATE TABLE linestring_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY linestring_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|LINESTRING(42.1 71, 41.4 70, 41.3 72.9, 42.99 71.46, 44.47 73.21)
>>\.
=> SELECT gid, ST_AsText(geog) FROM linestring_geog;
gid |          ST_AsText
```

```
-----+-----
 1 | LINESTRING (42.1 71, 41.4 70, 41.3 72.9, 42.99 71.46, 44.47 73.21)
(1 row)
```

Multilinestring

A collection of zero or more linestrings. A multilinestring has no dimension. The boundary of a multilinestring is a multipoint object containing the start and end points of all the linestrings.

The following are examples of multilinestrings:



Examples

The following example uses the GEOMETRY type to create a table, use copy to load a multilinestring to the table, and then queries the table to view the multilinestring:

```
=> CREATE TABLE multilinestring_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multilinestring_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|MULTILINESTRING((1 5, 2 4, 5 3, 6 6),(3 5, 3 7))
>>\.
=> SELECT gid, ST_AsText(geom) FROM multilinestring_geom;
gid | ST_AsText
-----+-----
 1 | MULTILINESTRING ((1 5, 2 4, 5 3, 6 6), (3 5, 3 7))
```

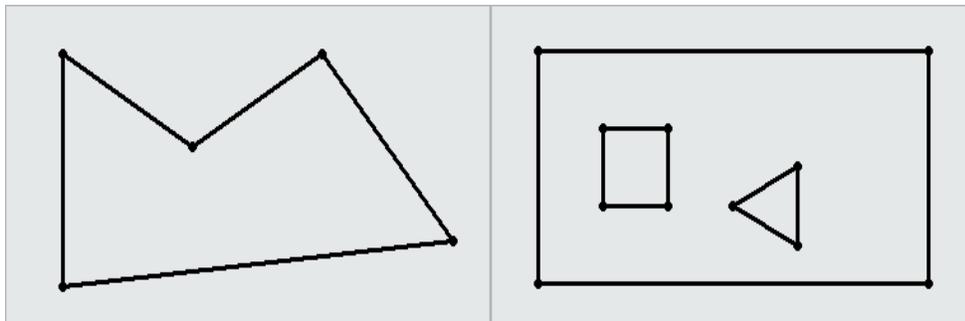
(1 row)

The following example uses the GEOGRAPHY type to create a table, use copy to load a multilinestring to the table, and then queries the table to view the multilinestring:

```
=> CREATE TABLE multilinestring_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY multilinestring_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM
stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|MULTILINESTRING((42.1 71, 41.4 70, 41.3 72.9), (42.99 71.46, 44.47 73.21))
>>\.
=> SELECT gid, ST_AsText(geog) FROM multilinestring_geog;
gid | ST_AsText
-----+-----
1 | MULTILINESTRING((42.1 71, 41.4 70, 41.3 72.9), (42.99 71.46, 44.47 73.21))
(1 row)
```

Polygon

An object identified by a set of closed linestrings. A polygon can have one or more holes, as defined by interior boundaries, but all points must be connected. Two examples of polygons are:



Inclusive and Exclusive Polygons

Polygons that include their points in clockwise order include all space inside the perimeter of the polygon and exclude all space outside that perimeter. Polygons that include their points in counterclockwise order exclude all space inside the perimeter and include all space outside that perimeter.

Examples

The following example uses the GEOMETRY type to create a table, use copy to load a polygon into the table, and then queries the table to view the polygon:

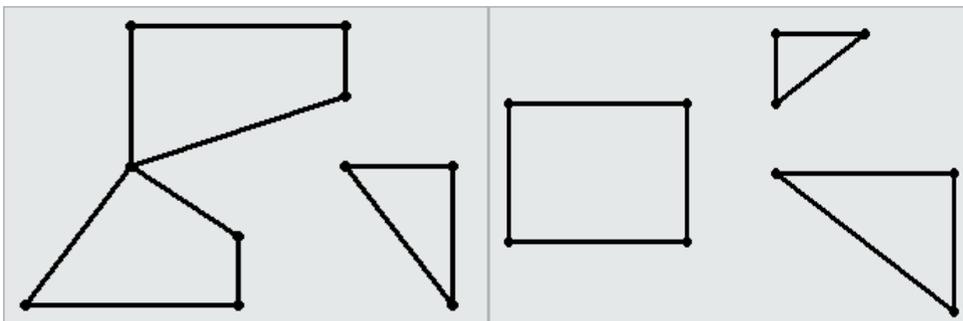
```
=> CREATE TABLE polygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY polygon_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter
'|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|POLYGON(( 2 6, 2 9, 6 9, 7 7, 4 6, 2 6))
>>\.
=> SELECT gid, ST_AsText(geom) FROM polygon_geom;
gid |          ST_AsText
-----+-----
  1 | POLYGON((2 6, 2 9, 6 9, 7 7, 4 6, 2 6))
(1 row)
```

The following example uses the GEOGRAPHY type to create a table, use copy to load a polygon into the table, and then queries the table to view the polygon:

```
=> CREATE TABLE polygon_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY polygon_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|POLYGON((42.1 71, 41.4 70, 41.3 72.9, 44.47 73.21, 42.99 71.46, 42.1 71))
>>\.
=> SELECT gid, ST_AsText(geog) FROM polygon_geog;
gid |          ST_AsText
-----+-----
  1 | POLYGON((42.1 71, 41.4 70, 41.3 72.9, 44.47 73.21, 42.99 71.46, 42.1 71))
(1 row)
```

Multipolygon

A collection of zero or more polygons that do not overlap.



Examples

The following example uses the GEOMETRY type to create a table, use copy to load a multipolygon into the table, and then queries the table to view the polygon:

```
=> CREATE TABLE multipolygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multipolygon_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>9|MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
>>\.
=> SELECT gid, ST_AsText(geom) FROM polygon_geom;
gid | ST_AsText
-----+-----
  9 | MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
(1 row)
```

The following example uses the GEOGRAPHY type to create a table, use copy to load a multipolygon into the table, and then queries the table to view the polygon:

```
=> CREATE TABLE multipolygon_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY polygon_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|POLYGON(((42.1 71, 41.4 70, 41.3 72.9, 44.47 73.21, 42.99 71.46, 42.1 71)))
>>\.
=> SELECT gid, ST_AsText(geog) FROM polygon_geog;
gid | ST_AsText
-----+-----
  1 | POLYGON(((42.1 71, 41.4 70, 41.3 72.9, 42.1 71)),((44.47 73.21, 42.99 71.46, 42.1 71, 44.47
73.21)))
(1 row)
```

Spatial Object Representations

The OGC defines two ways to represent spatial objects:

- Well-Known Text (WKT)
- Well-Known Binary (WKB)

Well-Known Text (WKT)

Well-Known Text (WKT) is an ASCII representation of a spatial object.

WKTs are not case sensitive; Vertica recognizes any combination of lowercase and uppercase letters.

Some examples of valid WKTs are:

WKT Example	Description
POINT(1 2)	The point (1,2)
MULTIPOINT(0 0,1 1)	A set made up of the points (0,0) and (1,1)
LINSTRING(1.5 2.45,3.21 4)	The line from the point (1.5,2.45) to the point (3.21,4)
MULTILINESTRING((0 0,-1 -2,-3 -4),(2 3,3 4,6 7))	Two linestrings, one that passes through (0,0), (-1,-2), and (-3,-4), and one that passes through (2,3), (3,4), and (6,7).
POLYGON((1 2,1 4,3 4,3 2,1 2))	The rectangle whose four corners are indicated by (1,2), (1,4), (3,4), and (3,2). A polygon must be closed, so the first and last points in the WKT must match.
POLYGON((0.5 0.5,5 0,5 0 5,0 5,0.5 0.5), (1.5 1,4 3,4 1,1.5 1))	A polygon (0.5 0.5,5 0,5 0 5,0.5 0.5) with a hole in it (1.5 1,4 3,4 1,1.5 1).
MULTIPOLYGON(((0 1,3 0,4 3,0 4,0 1)), ((3 4,6 3,5 5,3 4)), ((0 0,-1 -2,-3 -2,-2 -1,0 0)))	A set of three polygons
GEOMETRYCOLLECTION(POINT (5 8), LINSTRING(-1 3,1 4))	A set containing the point (5,8) and the line from (-1,3) to (1,4)
POINT EMPTY MULTIPOINT EMPTY LINSTRING EMPTY MULTILINESTRING EMPTY MULTILINESTRING(EMPTY) POLYGON EMPTY POLYGON(EMPTY) MULTIPOLYGON EMPTY MULTIPOLYGON(EMPTY)	Empty spatial objects; empty objects have no points.

Invalid WKTs are:

- POINT(1 NAN), POINT(1 INF)—Coordinates must be numbers.
- POLYGON((1 2, 1 4, 3 4, 3 2))—A polygon must be closed.
- POLYGON((1 4, 2 4))—A linestring is not a valid polygon.

Well-Known Binary (WKB)

Well-Known Binary (WKB) is a binary representation of a spatial object. This format is primarily used to port spatial data between applications.

Spatial Definitions

The OGC defines properties that describe

- Characteristics of spatial objects
- Spatial relationships that can exist among objects

Vertica provides functions that test for and analyze the following properties and relationships.

Boundary

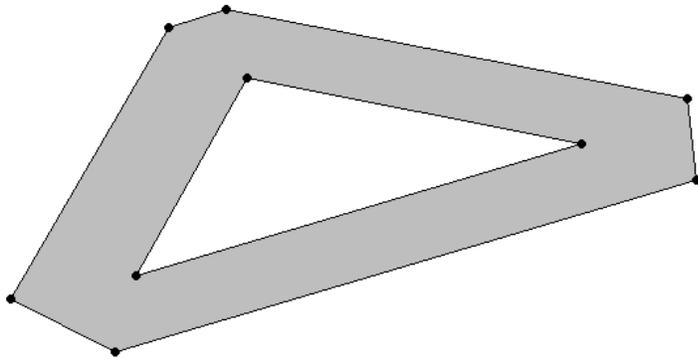
The set of points that define the limit of a spatial object:

- Points, multipoints, and geometrycollections do not have boundaries.
- The boundary of a linestring is a multipoint object. This object contains its start and end points.
- The boundary of a multilinestring is a multipoint object. This object contains the start and end points of all the linestrings that make up the multilinestring.
- The boundary of a polygon is a linestring that begins and ends at the same point. If the polygon has one or more holes, the boundary is a multilinestring that contains the boundaries of the exterior polygon and any interior polygons.
- The boundary of a multipolygon is a multilinestring that contains the boundaries of all the polygons that make up the multipolygon.

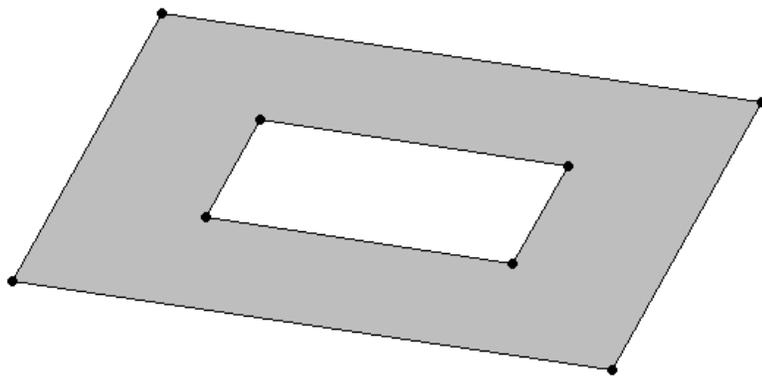
Buffer

The set of all points that are within or equal to a specified distance from the boundary of a spatial object. The distance can be positive or negative.

Positive buffer:



Negative buffer:



Contains

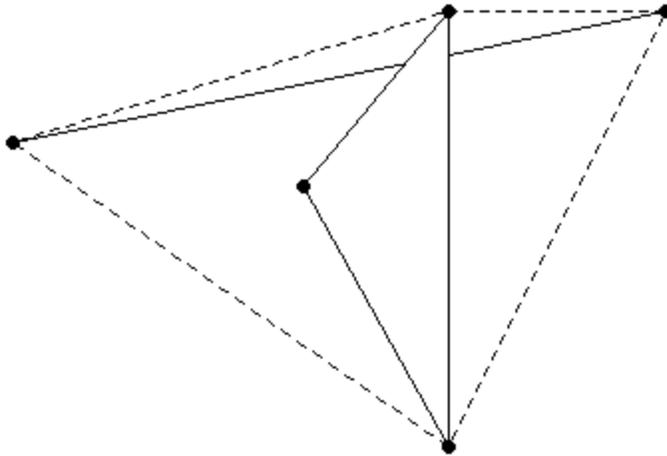
One spatial object contains another spatial object if its interior includes all points of the other object. If an object such as a point or linestring only exists along a polygon's boundary, the polygon does not contain it. If a point is on a linestring, the linestring contains it; the interior of a linestring is all the points on the linestring *except* the start and end points.

Contains(a, b) is spatially equivalent to within(b, a).

Convex Hull

The smallest convex polygon that contains one or more spatial objects.

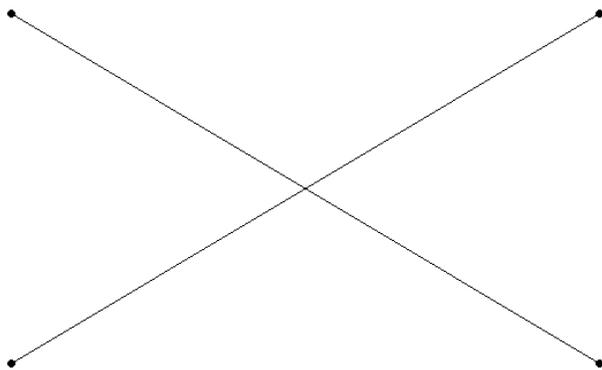
In the following figure, the dotted lines represent the convex hull for a linestring and a triangle.



Crosses

Two spatial objects cross if both of the following are true:

- The two objects have some but not all interior points in common.
- The dimension of the result of their intersection is less than the maximum dimension of the two objects.



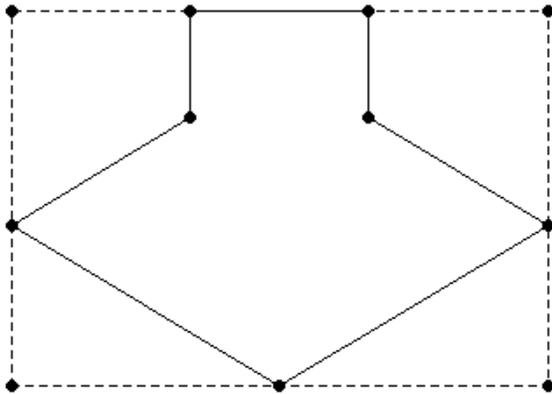
Disjoint

Two spatial objects have no points in common; they do not intersect or touch.

Envelope

The minimum bounding rectangle that contains a spatial object.

The envelope for the following polygon is represented by the dotted lines in the following figure.



Equals

Two spatial objects are equal when their coordinates match exactly. Synonymous with *spatially equivalent*.

The order of the points do not matter in determining spatial equivalence:

- `LINestring(1 2, 4 3)` equals `LINestring(4 3, 1 2)`.
- `POLYGON ((0 0, 1 1, 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0))` equals `POLYGON((1 1, 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0, 1 1))`.
- `MULTILINESTRING((1 2, 4 3),(0 0, -1 -4))` equals `MULTILINESTRING((0 0, -1 -4),(1 2, 4 3))`.

Exterior

The set of points not contained within a spatial object nor on its boundary.

GeometryCollection

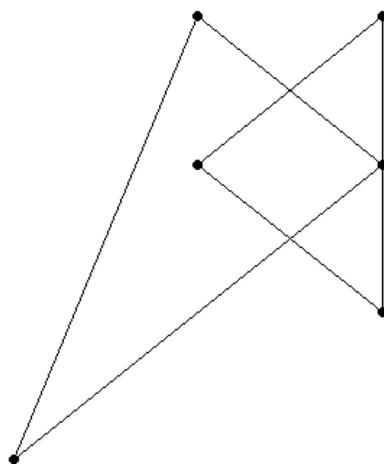
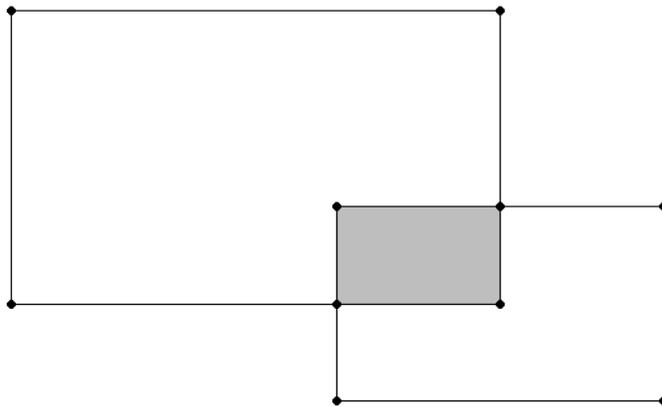
A set of zero or more objects from any of the supported classes of spatial objects.

Interior

The set of points contained in a spatial object, excluding its boundary.

Intersection

The set of points that two or more spatial objects have in common.



Overlaps

If a spatial object shares space with another object, but is not contained within that object, the objects overlap. The objects must overlap at their interiors; if two objects touch at a single point or intersect only along a boundary, they do not overlap.

Relates

When a spatial object is spatially related to another object as defined by a DE-9IM pattern matrix string.

A DE-9IM pattern matrix string identifies how two spatial objects are spatially related to each other. For more information about the DE-9IM standard, see [Understanding Spatial Relations](#).

Simple

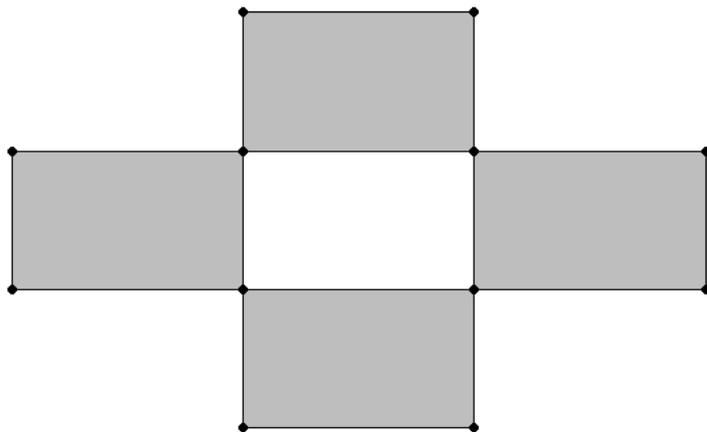
For points, multipoints, linestrings, or multilinestrings, a spatial object is simple if it does not intersect itself or has no self-tangency points.

Polygons, multipolygons, and geometrycollections are always simple.

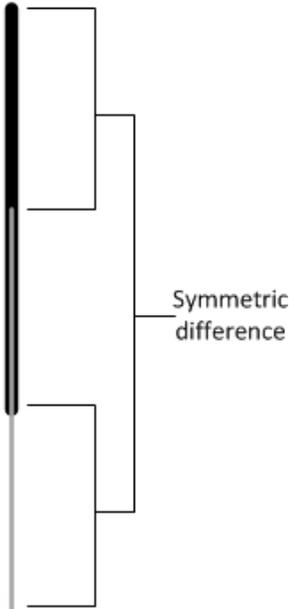
Symmetric Difference

The set of all points of a pair of spatial objects where the objects do not intersect. This difference is spatially equivalent to the union of the two objects less their intersection. The symmetric difference contains the boundaries of the intersections.

In the following figure, the shaded areas represent the symmetric difference of the two rectangles.

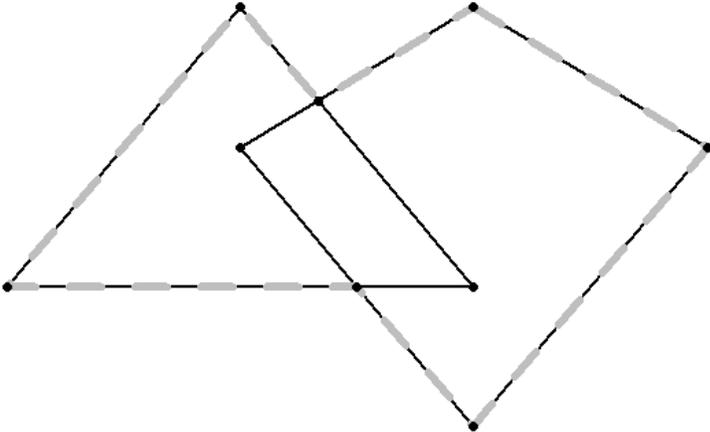


The following figure shows the symmetric difference of two overlapping linestrings.



Union

For two or more spatial objects, the set of all points in all the objects.



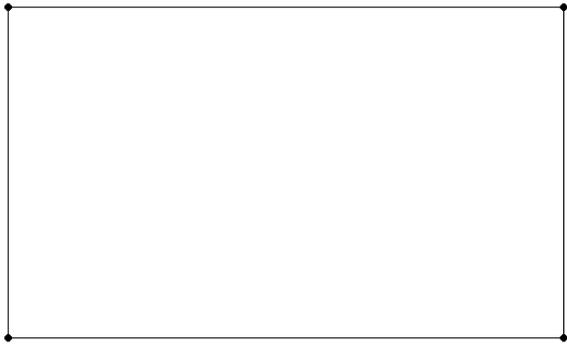
Validity

For a polygon or multipolygon, when all of the following are true:

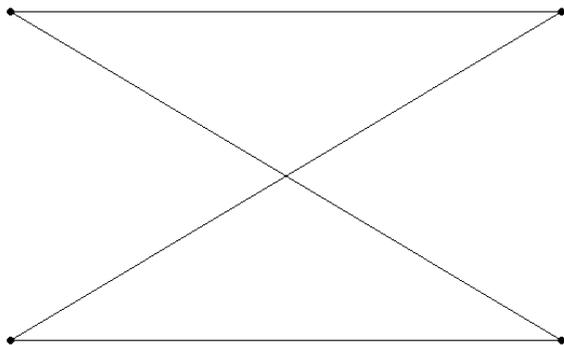
- It is closed; its start point is the same as its end point.
- Its boundary is a set of linestrings.

- No two linestrings in the boundary cross. The linestrings in the boundary may touch at a point but they cannot cross.
- Any polygons in the interior must be completely contained; they cannot touch the boundary of the exterior polygon *except* at a vertex.

Valid polygons:



Invalid polygon:

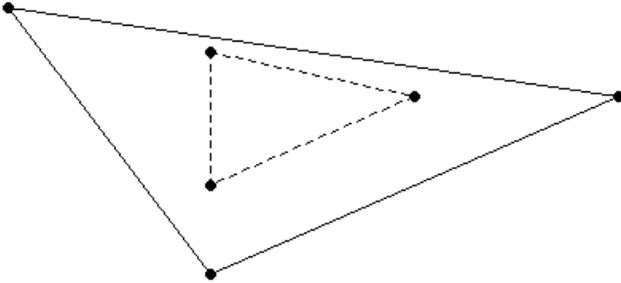


Within

A spatial object is considered within another spatial object when all its points are inside the other object's interior. Thus, if a point or linestring only exists along a polygon's boundary, it is not considered within the polygon. The polygon boundary is not part of its interior.

If a point is on a linestring, it is considered within the linestring. The interior of a linestring is all the points along the linestring, except the start and end points.

`Within(a, b)` is spatially equivalent to `contains(b, a)`.



Spatial Data Type Support Limitations

Vertica does not support all types of GEOMETRY and GEOGRAPHY objects. See the respective function page for a list of objects that function supports. Spherical geometry is generally more complex than Euclidean geometry. Thus, there are fewer spatial functions that support the GEOGRAPHY data type.

Limitations of spatial data type support:

- A non-WGS84 GEOGRAPHY object is a spatial object defined on the surface of a perfect sphere of radius 6371 kilometers. This sphere approximates the shape of the earth. Other spatial programs may use an ellipsoid to model the earth, resulting in slightly different data.
- You cannot modify the size or data type of a GEOMETRY or GEOGRAPHY column after creation.
- You cannot import data to or export data from tables that contain spatial data from another Vertica database.
- You can only use the STV_Intersect functions with points and polygons.
- GEOGRAPHY objects of type GEOMETRYCOLLECTION are not supported.
- Values for longitude must be between -180 and +180 degrees. Values for latitude must be between -90 and +90 degrees. The Vertica geospatial functions do not validate these values.
- GEOMETRYCOLLECTION objects cannot contain empty objects. For example, you cannot specify `GEOMETRYCOLLECTION (LINESTRING(1 2, 3 4), POINT(5 6), POINT EMPTY)`.
- If you pass a spatial function a NULL geometry, the function returns NULL, unless otherwise specified. A result of NULL has no value.

- Polymorphic functions, such as NVL and GREATEST, do not accept GEOMETRY and GEOGRAPHY arguments.

Time Series Analytics

Time series analytics evaluate the values of a given set of variables over time and group those values into a window (based on a time interval) for analysis and aggregation. Common scenarios for using time series analytics include: stock market trades and portfolio performance changes over time, and charting trend lines over data.

Since both time and the state of data within a time series are continuous, it can be challenging to evaluate SQL queries over time. Input records often occur at non-uniform intervals, which can create gaps. To solve this problem Vertica provides:

- Gap-filling functionality, which fills in missing data points
- Interpolation scheme, which constructs new data points within the range of a discrete set of known data points.

Vertica interpolates the non-time series columns in the data (such as analytic function results computed over time slices) and adds the missing data points to the output. This section describes gap filling and interpolation in detail.

You can use [event-based windows](#) to break time series data into windows that border on significant events within the data. This is especially relevant in financial data, where analysis might focus on specific events as triggers to other activity.

[Sessionization](#) is a special case of event-based windows that is frequently used to analyze click streams, such as identifying web browsing sessions from recorded web clicks.

Vertica provides additional support for time series analytics with the following SQL extensions:

- [TIMESERIES clause](#) in a SELECT statement supports gap-filling and interpolation (GFI) computation.
- [TS_FIRST_VALUE](#) and [TS_LAST_VALUE](#) are time series aggregate functions that return the value at the start or end of a time slice, respectively, which is determined by the interpolation scheme.
- [TIME_SLICE](#) is a (SQL extension) date/time function that aggregates data by different fixed-time intervals and returns a rounded-up input `TIMESTAMP` value to a value that corresponds with the start or end of the time slice interval.

See Also

- [SQL Analytics](#)
- [Event-Based Windows](#)
- [Sessionization with Event-Based Windows](#)

Gap Filling and Interpolation (GFI)

The examples and graphics that explain the concepts in this topic use the following simple schema:

```
CREATE TABLE TickStore (ts TIMESTAMP, symbol VARCHAR(8), bid FLOAT);
INSERT INTO TickStore VALUES ('2009-01-01 03:00:00', 'XYZ', 10.0);
INSERT INTO TickStore VALUES ('2009-01-01 03:00:05', 'XYZ', 10.5);
COMMIT;
```

In Vertica, time series data is represented by a sequence of rows that conforms to a particular table schema, where one of the columns stores the time information.

Both time and the state of data within a time series are continuous. Thus, evaluating SQL queries over time can be challenging because input records usually occur at non-uniform intervals and can contain gaps.

For example, the following table contains two input rows five seconds apart: 3:00:00 and 3:00:05.

```
=> SELECT * FROM TickStore;
      ts          | symbol | bid
-----+-----+----
2009-01-01 03:00:00 | XYZ   | 10
2009-01-01 03:00:05 | XYZ   | 10.5
(2 rows)
```

Given those two inputs, how can you determine a bid price that falls between the two points, such as at 3:00:03 PM?

The [TIME_SLICE](#) function normalizes timestamps into corresponding time slices; however, [TIME_SLICE](#) does not solve the problem of missing inputs (time slices) in the data. Instead, Vertica provides gap-filling and interpolation (GFI) functionality, which fills in missing data points and adds new (missing) data points within a range of known data points to the output. It accomplishes these tasks with [time series aggregate functions](#) and the SQL [TIMESERIES Clause](#).

But first, we'll illustrate the components that make up gap filling and interpolation in Vertica, starting with [Constant Interpolation](#).

The images in the following topics use the following legend:

- The x-axis represents the timestamp (ts) column
- The y-axis represents the bid column.
- The vertical blue lines delimit the time slices.

- The red dots represent the input records in the table, \$10.0 and \$10.5.
- The blue stars represent the output values, including interpolated values.

Constant Interpolation

Given known input timestamps at 03:00:00 and 03:00:05 in the [sample TickStore schema](#), how might you determine the bid price at 03:00:03?

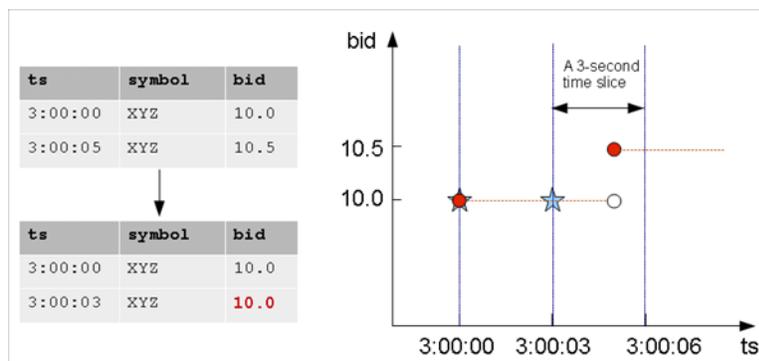
A common interpolation scheme used on financial data is to set the bid price to *the last seen value so far*. This scheme is referred to as **constant interpolation**, in which Vertica computes a new value based on the previous input records.

Note: Constant is Vertica's default interpolation scheme. Another interpolation scheme, [linear](#), is discussed in an upcoming topic.

Returning to the problem query, here is the table output, which shows a 5-second lag between bids at 03:00:00 and 03:00:05:

```
=> SELECT * FROM TickStore;
      ts          | symbol | bid
-----+-----+----
2009-01-01 03:00:00 | XYZ   | 10
2009-01-01 03:00:05 | XYZ   | 10.5
(2 rows)
```

Using constant interpolation, the interpolated bid price of XYZ remains at \$10.0 at 3:00:03, which falls between the two known data inputs (3:00:00 PM and 3:00:05). At 3:00:05, the value changes to \$10.5. The known data points are represented by a red dot, and the interpolated value at 3:00:03 is represented by the blue star.



In order to write a query that makes the input rows more uniform, you first need to understand the [TIMESERIES clause and time series aggregate functions](#).

TIMESERIES Clause and Aggregates

The SELECT..TIMESERIES clause and time series aggregates help solve the problem of gaps in input records by normalizing the data into 3-second time slices and interpolating the bid price when it finds gaps.

TIMESERIES Clause

The [TIMESERIES Clause](#) is an important component of time series analytics computation. It performs gap filling and interpolation (GFI) to generate time slices missing from the input records. The clause applies to the timestamp columns/expressions in the data, and takes the following form:

```
TIMESERIES slice_time AS 'length_and_time_unit_expression'  
OVER ( ... [ window-partition-clause[ , ... ] ]  
... ORDER BY time_expression )  
... [ ORDER BY table_column [ , ... ] ]
```

Note: The TIMESERIES clause requires an ORDER BY operation on the timestamp column.

Time Series Aggregate Functions

[Timeseries Aggregate \(TSA\) functions](#) evaluate the values of a given set of variables over time and group those values into a window for analysis and aggregation.

TSA functions process the data that belongs to each time slice. One output row is produced per time slice or per partition per time slice if a partition expression is present.

The following table shows 3-second time slices where:

- The first two rows fall within the first time slice, which runs from 3:00:00 to 3:00:02. These are the input rows for the TSA function's output for the time slice starting at 3:00:00.
- The second two rows fall within the second time slice, which runs from 3:00:03 to 3:00:05. These are the input rows for the TSA function's output for the time slice starting at 3:00:03.

The result is the start of each time slice.

ts	symbol	bid		ts	symbol	bid
3:00:00	XYZ	10.0		3:00:00	XYZ	10.0
3:00:01	XYZ	10.1		3:00:03	XYZ	10.1
3:00:04	XYZ	10.3				
3:00:05	XYZ	10.5				

Examples

The following examples compare the values returned with and without the `TS_FIRST_VALUE` TSA function.

This example shows the `TIMESERIES` clause without the `TS_FIRST_VALUE` TSA function.

```
=> SELECT slice_time, bid FROM TickStore TIMESERIES slice_time AS '3 seconds' OVER(PARTITION by TickStore.bid ORDER BY ts);
```

This example shows both the `TIMESERIES` clause and the `TS_FIRST_VALUE` TSA function. The query returns the values of the `bid` column, as determined by the specified constant interpolation scheme.

```
=> SELECT slice_time, TS_FIRST_VALUE(bid, 'CONST') bid FROM TickStore TIMESERIES slice_time AS '3 seconds' OVER(PARTITION by symbol ORDER BY ts);
```

Vertica interpolates the last known value and fills in the missing datapoint, returning 10 at 3:00:03:

First query		Interpolated value
<pre> slice_time bid -----+----- 2009-01-01 03:00:00 10 2009-01-01 03:00:03 10.5 (2 rows) </pre>	==>	<pre> slice_time bid -----+----- 2009-01-01 03:00:00 10 2009-01-01 03:00:03 10 (2 rows) </pre>

Time Series Rounding

Vertica calculates all time series as equal intervals relative to the timestamp `2000-01-01 00:00:00`. Vertica rounds time series timestamps as needed, to conform with this baseline. Start times are also rounded down to the nearest whole unit for the specified interval.

Given this logic, the `TIMESERIES` clause generates series of timestamps as described in the following sections.

Minutes

Time series of minutes are rounded down to full minutes. For example, the following statement specifies a time span of 00:00:03 - 00:05:50:

```
=> SELECT ts FROM (  
  SELECT '2015-01-04 00:00:03'::TIMESTAMP AS tm  
  UNION  
  SELECT '2015-01-04 00:05:50'::TIMESTAMP AS tm  
) t  
TIMESERIES ts AS '1 minute' OVER (ORDER BY tm);
```

Vertica rounds down the time series start and end times to full minutes, 00:00:00 and 00:05:00, respectively:

```
      ts  
-----  
2015-01-04 00:00:00  
2015-01-04 00:01:00  
2015-01-04 00:02:00  
2015-01-04 00:03:00  
2015-01-04 00:04:00  
2015-01-04 00:05:00  
(6 rows)
```

Weeks

Because the baseline timestamp 2000-01-01 00:00:00 is a Saturday, all time series of weeks start on Saturday. Vertica rounds down the series start and end timestamps accordingly. For example, the following statement specifies a time span of 12/10/99 - 01/10/00:

```
=> SELECT ts FROM (  
  SELECT '1999-12-10 00:00:00'::TIMESTAMP AS tm  
  UNION  
  SELECT '2000-01-10 23:59:59'::TIMESTAMP AS tm  
) t  
TIMESERIES ts AS '1 week' OVER (ORDER BY tm);
```

The specified time span starts on Friday (12/10/99), so Vertica starts the time series on the preceding Saturday, 12/04/99. The time series ends on the last Saturday within the time span, 01/08/00:

```
      ts  
-----  
1999-12-04 00:00:00  
1999-12-11 00:00:00  
1999-12-18 00:00:00
```

```
1999-12-25 00:00:00
2000-01-01 00:00:00
2000-01-08 00:00:00
(6 rows)
```

Months

Time series of months are divided into equal 30-day intervals, relative to the baseline timestamp `2000-01-01 00:00:00`. For example, the following statement specifies a time span of `09/01/99 - 12/31/00`:

```
=> SELECT ts FROM (
  SELECT '1999-09-01 00:00:00'::TIMESTAMP AS tm
  UNION
  SELECT '2000-12-31 23:59:59'::TIMESTAMP AS tm
) t
TIMESERIES ts AS '1 month' OVER (ORDER BY tm);
```

Vertica generates a series of 30-day intervals, where each timestamp is rounded up or down, relative to the baseline timestamp:

```
      ts
-----
1999-08-04 00:00:00
1999-09-03 00:00:00
1999-10-03 00:00:00
1999-11-02 00:00:00
1999-12-02 00:00:00
2000-01-01 00:00:00
2000-01-31 00:00:00
2000-03-01 00:00:00
2000-03-31 00:00:00
2000-04-30 00:00:00
2000-05-30 00:00:00
2000-06-29 00:00:00
2000-07-29 00:00:00
2000-08-28 00:00:00
2000-09-27 00:00:00
2000-10-27 00:00:00
2000-11-26 00:00:00
2000-12-26 00:00:00
(18 rows)
```

Years

Time series of years are divided into equal 365-day intervals. If a time span overlaps leap years since or before the baseline timestamp `2000-01-01 00:00:00`, Vertica rounds the series timestamps accordingly.

For example, the following statement specifies a time span of 01/01/95 - 05/08/09, which overlaps four leap years, including the baseline timestamp:

```
=> SELECT ts FROM (  
    SELECT '1995-01-01 00:00:00'::TIMESTAMP AS tm  
    UNION  
    SELECT '2009-05-08'::TIMESTAMP AS tm  
  ) t timeseries ts AS '1 year' over (ORDER BY tm);
```

Vertica generates a series of timestamps that are rounded up or down, relative to the baseline timestamp:

```
      ts  
-----  
1994-01-02 00:00:00  
1995-01-02 00:00:00  
1996-01-02 00:00:00  
1997-01-01 00:00:00  
1998-01-01 00:00:00  
1999-01-01 00:00:00  
2000-01-01 00:00:00  
2000-12-31 00:00:00  
2001-12-31 00:00:00  
2002-12-31 00:00:00  
2003-12-31 00:00:00  
2004-12-30 00:00:00  
2005-12-30 00:00:00  
2006-12-30 00:00:00  
2007-12-30 00:00:00  
2008-12-29 00:00:00  
(16 rows)
```

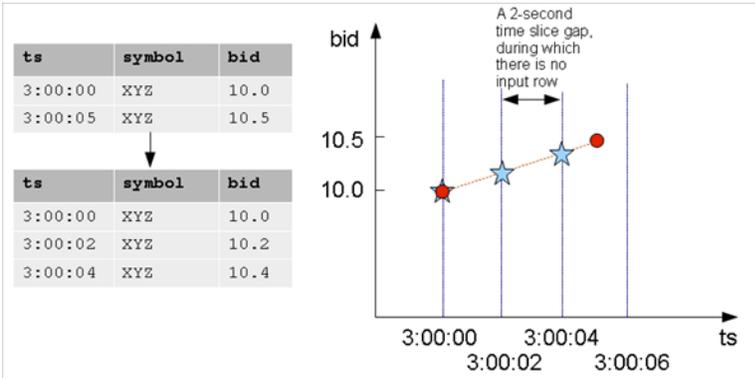
Linear Interpolation

Instead of interpolating data points based on the last seen value ([Constant Interpolation](#)), linear interpolation is where Vertica interpolates values in a linear slope based on the specified time slice.

The query that follows uses linear interpolation to place the input records in 2-second time slices and return the first bid value for each symbol/time slice combination (the value at the start of the time slice):

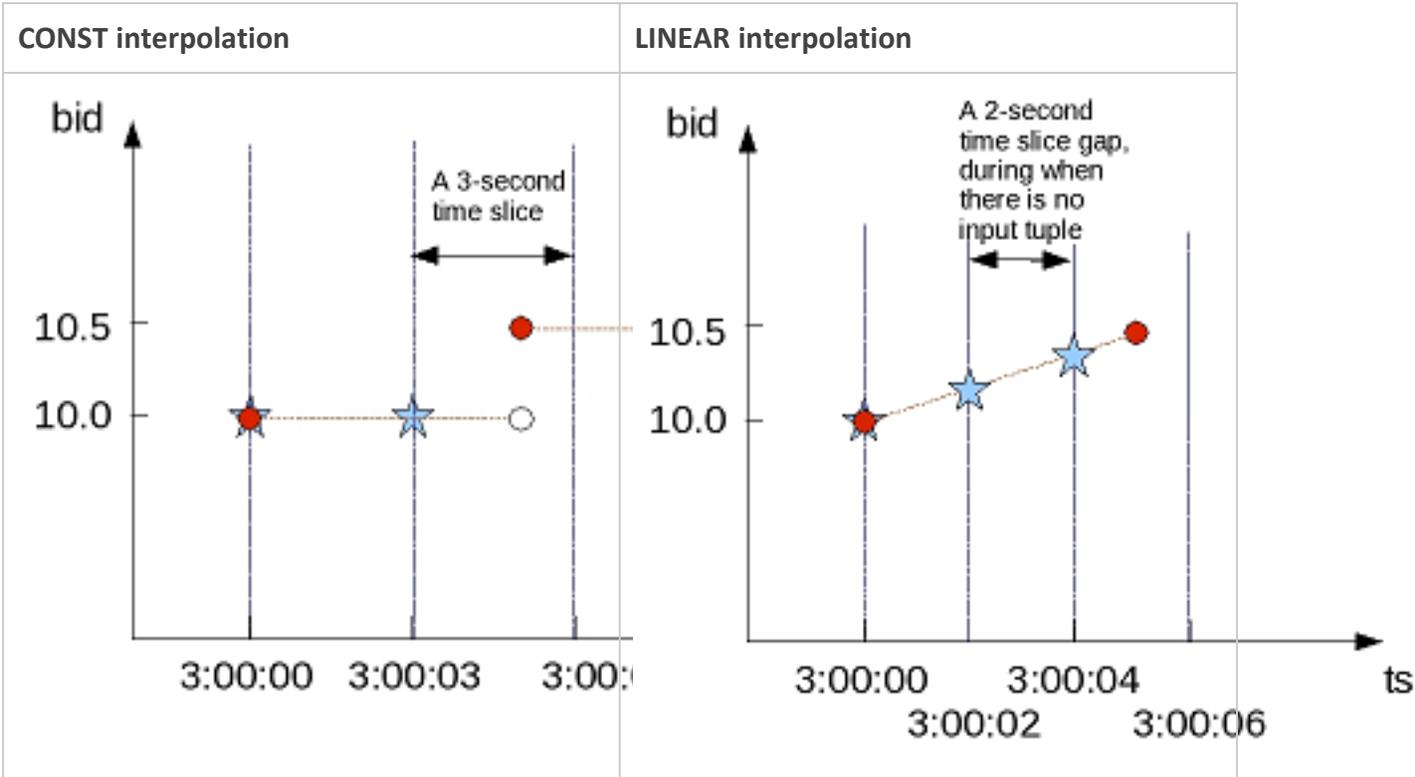
```
=> SELECT slice_time, TS_FIRST_VALUE(bid, 'LINEAR') bid FROM Tickstore  
    TIMESERIES slice_time AS '2 seconds' OVER(PARTITION BY symbol ORDER BY ts);  
    slice_time      | bid  
-----+-----  
2009-01-01 03:00:00 | 10  
2009-01-01 03:00:02 | 10.2  
2009-01-01 03:00:04 | 10.4  
(3 rows)
```

The following figure illustrates the previous query results, showing the 2-second time gaps (3:00:02 and 3:00:04) in which no input record occurs. Note that the interpolated bid price of XYZ changes to 10.2 at 3:00:02 and 10.3 at 3:00:03 and 10.4 at 3:00:04, all of which fall between the two known data inputs (3:00:00 and 3:00:05). At 3:00:05, the value would change to 10.5.



Note: The known data points above are represented by a red dot, and the interpolated values are represented by blue stars.

The following is a side-by-side comparison of constant and linear interpolation schemes.



GFI Examples

This topic illustrates some of the queries you can write using the constant and linear interpolation schemes.

Constant Interpolation

The first query uses `TS_FIRST_VALUE()` and the `TIMESERIES Clause` to place the input records in 3-second time slices and return the first bid value for each symbol/time slice combination (the value at the start of the time slice).

Note: The `TIMESERIES` clause requires an `ORDER BY` operation on the `TIMESTAMP` column.

```
=> SELECT slice_time, symbol, TS_FIRST_VALUE(bid) AS first_bid FROM TickStore
      TIMESERIES slice_time AS '3 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

Because the bid price of stock XYZ is 10.0 at 3:00:03, the `first_bid` value of the second time slice, which starts at 3:00:03 is till 10.0 (instead of 10.5) because the input value of 10.5 does not occur until 3:00:05. In this case, the interpolated value is inferred from the last value seen on stock XYZ for time 3:00:03:

slice_time	symbol	first_bid
2009-01-01 03:00:00	XYZ	10
2009-01-01 03:00:03	XYZ	10

(2 rows)

The next example places the input records in 2-second time slices to return the first bid value for each symbol/time slice combination:

```
=> SELECT slice_time, symbol, TS_FIRST_VALUE(bid) AS first_bid FROM TickStore
      TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

The result now contains three records in 2-second increments, all of which occur between the first input row at 03:00:00 and the second input row at 3:00:05. Note that the second and third output record correspond to a time slice where there is no input record:

slice_time	symbol	first_bid
2009-01-01 03:00:00	XYZ	10
2009-01-01 03:00:02	XYZ	10
2009-01-01 03:00:04	XYZ	10

(3 rows)

Using the same table schema, the next query uses `TS_LAST_VALUE()`, with the `TIMESERIES` clause to return the last values of each time slice (the values at the end of the time slices).

Note: Time series aggregate functions process the data that belongs to each time slice. One output row is produced per time slice or per partition per time slice if a partition expression is present.

```
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid) AS last_bid FROM TickStore
      TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

Notice that the last value output row is 10.5 because the value 10.5 at time 3:00:05 was the last point inside the 2-second time slice that started at 3:00:04:

slice_time	symbol	last_bid
2009-01-01 03:00:00	XYZ	10
2009-01-01 03:00:02	XYZ	10
2009-01-01 03:00:04	XYZ	10.5

(3 rows)

Remember that because constant interpolation is the default, the same results are returned if you write the query using the `CONST` parameter as follows:

```
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid, 'CONST') AS last_bid FROM TickStore
      TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

Linear Interpolation

Based on the same input records described in the constant interpolation examples, which specify 2-second time slices, the result of `TS_LAST_VALUE` with linear interpolation is as follows:

```
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid, 'linear') AS last_bid FROM TickStore
      TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

In the results, no `last_bid` value is returned for the last row because the query specified `TS_LAST_VALUE`, and there is no data point after the 3:00:04 time slice to interpolate.

slice_time	symbol	last_bid
2009-01-01 03:00:00	XYZ	10.2
2009-01-01 03:00:02	XYZ	10.4
2009-01-01 03:00:04	XYZ	

(3 rows)

Using Multiple Time Series Aggregate Functions

Multiple time series aggregate functions can exist in the same query. They share the same *gap-filling* policy as defined in the `TIMESERIES` clause; however, each time series aggregate function can specify its own interpolation policy. In the following example, there are two constant and one linear interpolation schemes, but all three functions use a three-second time slice:

```
=> SELECT slice_time, symbol,
        TS_FIRST_VALUE(bid, 'const') fv_c,
        TS_FIRST_VALUE(bid, 'linear') fv_l,
        TS_LAST_VALUE(bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS '3 seconds' OVER(PARTITION BY symbol ORDER BY ts);
```

In the following output, the original output is compared to output returned by multiple time series aggregate functions.

ts	symbol	bid	==>	slice_time	symbol	fv_c	fv_l	lv_c
03:00:00	XYZ	10		2009-01-01 03:00:00	XYZ	10	10	10
03:00:05	XYZ	10.5		2009-01-01 03:00:03	XYZ	10	10.3	10.5
(2 rows)				(2 rows)				

Using the Analytic LAST_VALUE Function

Here's an example using `LAST_VALUE()`, so you can see the difference between it and the GFI syntax.

```
=> SELECT *, LAST_VALUE(bid) OVER(PARTITION by symbol ORDER BY ts)
AS "last bid" FROM TickStore;
```

There is no gap filling and interpolation to the output values.

```
ts | symbol | bid | last bid
-----+-----+-----+-----
2009-01-01 03:00:00 | XYZ | 10 | 10
2009-01-01 03:00:05 | XYZ | 10.5 | 10.5
(2 rows)
```

Using slice_time

In a `TIMESERIES` query, you cannot use the column `slice_time` in the `WHERE` clause because the `WHERE` clause is evaluated before the `TIMESERIES` clause, and the `slice_time` column is

not generated until the `TIMESERIES` clause is evaluated. For example, Vertica does not support the following query:

```
=> SELECT symbol, slice_time, TS_FIRST_VALUE(bid IGNORE NULLS) AS fv
FROM TickStore
WHERE slice_time = '2009-01-01 03:00:00'
TIMESERIES slice_time as '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
ERROR: Time Series timestamp alias/Time Series Aggregate Functions not allowed in WHERE clause
```

Instead, you could write a subquery and put the predicate on `slice_time` in the outer query:

```
=> SELECT * FROM (
  SELECT symbol, slice_time,
    TS_FIRST_VALUE(bid IGNORE NULLS) AS fv
  FROM TickStore
  TIMESERIES slice_time AS '2 seconds'
  OVER (PARTITION BY symbol ORDER BY ts) ) sq
WHERE slice_time = '2009-01-01 03:00:00';
symbol | slice_time | fv
-----+-----+-----
XYZ | 2009-01-01 03:00:00 | 10
(1 row)
```

Creating a Dense Time Series

The `TIMESERIES` clause provides a convenient way to create a dense time series for use in an outer join with fact data. The results represent every time point, rather than just the time points for which data exists.

The examples that follow use the same `TickStore` schema described in [Gap Filling and Interpolation \(GFI\)](#), along with the addition of a new inner table for the purpose of creating a join:

```
=> CREATE TABLE inner_table (
  ts TIMESTAMP,
  bid FLOAT
);
=> CREATE PROJECTION inner_p (ts, bid) as SELECT * FROM inner_table
ORDER BY ts, bid UNSEGMENTED ALL NODES;
=> INSERT INTO inner_table VALUES ('2009-01-01 03:00:02', 1);
=> INSERT INTO inner_table VALUES ('2009-01-01 03:00:04', 2);
=> COMMIT;
```

You can create a simple union between the start and end range of the timeframe of interest in order to return every time point. This example uses a 1-second time slice:

```
=> SELECT ts FROM (
  SELECT '2009-01-01 03:00:00'::TIMESTAMP AS time FROM TickStore
  UNION
  SELECT '2009-01-01 03:00:05'::TIMESTAMP FROM TickStore) t
TIMESERIES ts AS '1 seconds' OVER(ORDER BY time);
```

```

      ts
-----
2009-01-01 03:00:00
2009-01-01 03:00:01
2009-01-01 03:00:02
2009-01-01 03:00:03
2009-01-01 03:00:04
2009-01-01 03:00:05
(6 rows)

```

The next query creates a union between the start and end range of the timeframe using 500-millisecond time slices:

```

=> SELECT ts FROM (
      SELECT '2009-01-01 03:00:00'::TIMESTAMP AS time
      FROM TickStore
      UNION
      SELECT '2009-01-01 03:00:05'::TIMESTAMP FROM TickStore) t
      TIMESERIES ts AS '500 milliseconds' OVER(ORDER BY time);
      ts
-----
2009-01-01 03:00:00
2009-01-01 03:00:00.5
2009-01-01 03:00:01
2009-01-01 03:00:01.5
2009-01-01 03:00:02
2009-01-01 03:00:02.5
2009-01-01 03:00:03
2009-01-01 03:00:03.5
2009-01-01 03:00:04
2009-01-01 03:00:04.5
2009-01-01 03:00:05
(11 rows)

```

The following query creates a union between the start- and end-range of the timeframe of interest using 1-second time slices:

```

=> SELECT * FROM (
      SELECT ts FROM (
        SELECT '2009-01-01 03:00:00'::timestamp AS time FROM TickStore
        UNION
        SELECT '2009-01-01 03:00:05'::timestamp FROM TickStore) t
        TIMESERIES ts AS '1 seconds' OVER(ORDER BY time) ) AS outer_table
      LEFT OUTER JOIN inner_table ON outer_table.ts = inner_table.ts;

```

The union returns a complete set of records from the left-joined table with the matched records in the right-joined table. Where the query found no match, it extends the right side column with null values:

```

      ts          |          ts          | bid
-----+-----+-----
2009-01-01 03:00:00 |          |
2009-01-01 03:00:01 |          |
2009-01-01 03:00:02 | 2009-01-01 03:00:02 | 1

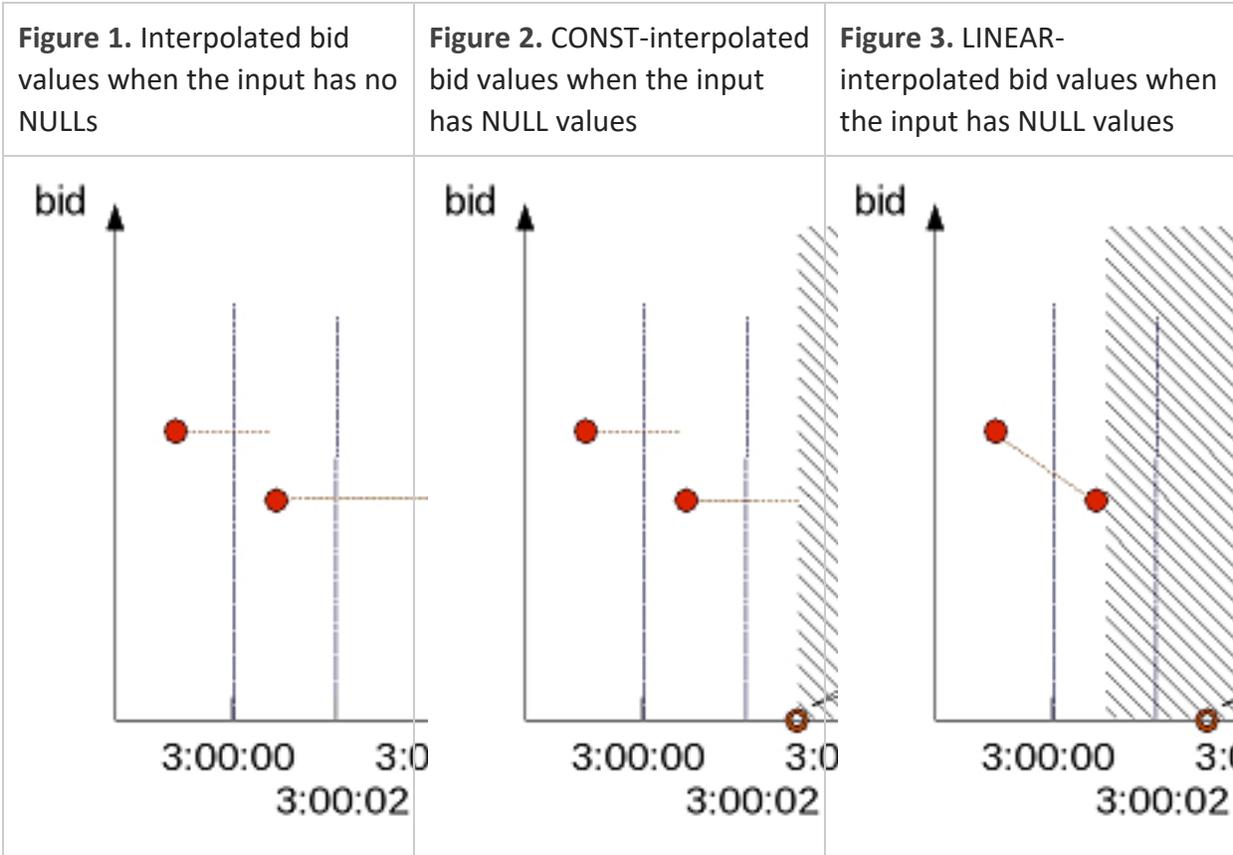
```

```
2009-01-01 03:00:03 | | |  
2009-01-01 03:00:04 | 2009-01-01 03:00:04 | 2  
2009-01-01 03:00:05 | | |  
(6 rows)
```

Null Values in Time Series Data

Null values are not common inputs for gap-filling and interpolation (GFI) computation, but if null values do exist, you can use time series aggregate functions (TS_FIRST_VALUE/TS_LAST_VALUE) with the IGNORE NULLS arguments to affect output of the interpolated values. The TSA functions are treated similarly to their analytic counterparts (FIRST_VALUE/LAST_VALUE) in that if the timestamp itself is null Vertica filter out those rows before gap filling and interpolation occurs.

The three images below will illustrate the points that follow on how Vertica handles time series data that contains null values.



Constant Interpolation with Null Values

Figure 1 illustrates a default (constant) interpolation result on four input rows where none of the inputs contains a NULL value. Figure 2 shows the same input rows with the addition of another input record whose bid value is NULL, and whose timestamp (ts) value is 3:00:03.

For constant interpolation, the bid value starting at 3:00:03 is null until the next non-null bid value appears in time. In Figure 2, the presence of the null row makes the interpolated bid value null in the time interval denoted by the shaded region. As a result, if `TS_FIRST_VALUE (bid)` is evaluated with constant interpolation on the time slice that begins at 3:00:02, its output is non-null. However, `TS_FIRST_VALUE (bid)` on the next time slice produces null. If the last value of the 3:00:02 time slice is null, the first value for the next time slice (3:00:04) is null. However, if you were to use a TSA function with `IGNORE NULLS`, then the value at 3:00:04 would be the same value as it was at 3:00:02.

To illustrate, insert a new row into the TickStore table at 03:00:03 with a null bid value, Vertica will output a row for the 03:00:02 record with a null value but no row for the 03:00:03 input:

```
=> INSERT INTO tickstore VALUES('2009-01-01 03:00:03', 'XYZ', NULL);
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid) AS last_bid FROM TickStore
-> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
  slice_time      | symbol | last_bid
-----+-----+-----
2009-01-01 03:00:00 | XYZ   |      10
2009-01-01 03:00:02 | XYZ   |
2009-01-01 03:00:04 | XYZ   |     10.5
(3 rows)
```

If you specify `IGNORE NULLS`, Vertica fills in the missing data point using a constant interpolation scheme. Here, the bid price at 03:00:02 is interpolated to the last known input record for bid, which was \$10 at 03:00:00:

```
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid IGNORE NULLS) AS last_bid FROM TickStore
-> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
  slice_time      | symbol | last_bid
-----+-----+-----
2009-01-01 03:00:00 | XYZ   |      10
2009-01-01 03:00:02 | XYZ   |      10
2009-01-01 03:00:04 | XYZ   |     10.5
(3 rows)
```

Now if you were to insert a row where the timestamp column contained a null value, Vertica would filter out that row before gap filling and interpolation occurred.

```
=> INSERT INTO tickstore VALUES(NULL, 'XYZ', 11.2);
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid) AS last_bid FROM TickStore
-> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

Notice there is no output for the 11.2 bid row:

```
      slice_time      | symbol | last_bid
-----+-----+-----
2009-01-01 03:00:00 | XYZ   |      10
2009-01-01 03:00:02 | XYZ   |
2009-01-01 03:00:04 | XYZ   |     10.5
(3 rows)
```

Linear Interpolation with Null Values

For linear interpolation, the interpolated bid value becomes null in the time interval, which is represented by the shaded region in Figure 3. In the presence of an input null value at 3:00:03, Vertica cannot linearly interpolate the bid value around that time point.

Vertica takes the closest non null value on either side of the time slice and uses that value. For example, if you use a linear interpolation scheme and you do not specify IGNORE NULLS, and your data has one real value and one null, the result is null. If the value on either side is null, the result is null. Therefore, to evaluate `TS_FIRST_VALUE(bid)` with linear interpolation on the time slice that begins at 3:00:02, its output is null. `TS_FIRST_VALUE(bid)` on the next time slice remains null.

```
=> SELECT slice_time, symbol, TS_FIRST_VALUE(bid, 'linear') AS fv_1 FROM TickStore
-> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
      slice_time      | symbol | fv_1
-----+-----+-----
2009-01-01 03:00:00 | XYZ   |      10
2009-01-01 03:00:02 | XYZ   |
2009-01-01 03:00:04 | XYZ   |
(3 rows)
```

Data Aggregation

You can use functions such as [SUM](#) and [COUNT](#) to aggregate the results of `GROUP BY` queries at one or more levels.

Aggregating Data at a Single Level

The simplest `GROUP BY` queries aggregate data at a single level. For example, a table might contain the following information about family expenses:

- Category
- Amount spent on that category during the year
- Year

Table data might look like this:

```
=> SELECT * FROM expenses ORDER BY Category;
Year | Category | Amount
-----+-----+-----
2005 | Books    | 39.98
2007 | Books    | 29.99
2008 | Books    | 29.99
2006 | Electrical | 109.99
2005 | Electrical | 109.99
2007 | Electrical | 229.98
```

You can use aggregate functions to get the total expenses per category or per year:

```
=> SELECT SUM(Amount), Category FROM expenses GROUP BY Category;
SUM      | Category
-----+-----
 99.96   | Books
449.96   | Electrical

=> SELECT SUM(Amount), Year FROM expenses GROUP BY Year;
SUM      | Year
-----+-----
149.97   | 2005
109.99   | 2006
 29.99   | 2008
259.97   | 2007
```

Aggregating Data at Multiple Levels

Over time, tables that are updated frequently can contain large amounts of data. Using the simple table shown earlier, suppose you want a multilevel query, like the number of expenses per category per year.

The following query uses the ROLLUP aggregation with the SUM function to calculate the total expenses by category and the overall expenses total. The NULL fields indicate subtotal values in the aggregation.

- When only the Year column is NULL, the subtotal is for all the Category values.
- When both the Year and Category columns are NULL, the subtotal is for all Amount values for both columns.

Using the ORDER BY clause orders the results by expense category, the year the expenses took place, and the GROUP BY level that the GROUPING_ID function creates:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Books    |      | 99.96
Electrical | 2005 | 109.99
Electrical | 2006 | 109.99
Electrical | 2007 | 229.98
Electrical |      | 449.96
         |      | 549.92
```

Similarly, the following query calculates the total sales by year and the overall sales total and then uses the ORDER BY clause to sort the results:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY ROLLUP(Year, Category) ORDER BY 2, 1, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Electrical | 2005 | 109.99
         | 2005 | 149.97
Electrical | 2006 | 109.99
         | 2006 | 109.99
Books    | 2007 | 29.99
Electrical | 2007 | 229.98
         | 2007 | 259.97
Books    | 2008 | 29.99
```

```
      | 2008 | 29.99  
      |      | 549.92  
(11 rows)
```

You can use the CUBE aggregate to perform all possible groupings of the category and year expenses. The following query returns all possible groupings, ordered by grouping:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses  
      GROUP BY CUBE(Category, Year) ORDER BY 1, 2, GROUPING_ID();  
Category | Year | SUM  
-----+-----+-----  
Books    | 2005 | 39.98  
Books    | 2007 | 29.99  
Books    | 2008 | 29.99  
Books    |      | 99.96  
Electrical | 2005 | 109.99  
Electrical | 2006 | 109.99  
Electrical | 2007 | 229.98  
Electrical |      | 449.96  
          | 2005 | 149.97  
          | 2006 | 109.99  
          | 2007 | 259.97  
          | 2008 | 29.99  
          |      | 549.92
```

The results include subtotals for each category and each year and a total (\$549.92) for all transactions, regardless of year or category.

ROLLUP, CUBE, and GROUPING SETS generate NULL values in grouping columns to identify subtotals. If table data includes NULL values, differentiating these from NULL values in subtotals can sometimes be challenging.

In the preceding output, the NULL values in the Year column indicate that the row was grouped on the Category column, rather than on both columns. In this case, ROLLUP added the NULL value to indicate the subtotal row.

To distinguish subtotal rows from NULL values that are part of the input data, use the [GROUPING](#) function.

Aggregate Expressions for GROUP BY

You can include CUBE and ROLLUP aggregates within a GROUPING SETS aggregate. Be aware that the CUBE and ROLLUP aggregates can result in a large amount of output. However, you can avoid large outputs by using GROUPING SETS to return only specified results.

```
...GROUP BY a,b,c,d,ROLLUP(a,b)...  
...GROUP BY a,b,c,d,CUBE((a,b),c,d)...
```

You cannot include any aggregates in a CUBE or ROLLUP aggregate expression.

You can append multiple GROUPING SETS, CUBE, or ROLLUP aggregates in the same query.

```
...GROUP BY a,b,c,d,CUBE(a,b),ROLLUP (c,d)...  
...GROUP BY a,b,c,d,GROUPING SETS ((a,d),(b,c),CUBE(a,b));...  
...GROUP BY a,b,c,d,GROUPING SETS ((a,d),(b,c),(a,b),(a),(b),())...
```

Aggregates and Functions for Multilevel Grouping

Vertica provides several aggregates and functions that group the results of a GROUP BY query at multiple levels.

Aggregates for Multilevel Grouping

Use the following aggregates for multilevel grouping:

- **ROLLUP** automatically performs subtotal aggregations. ROLLUP performs one or more aggregations across multiple dimensions, at different levels.
- **CUBE** performs the aggregation for all permutations of the CUBE expression that you specify.
- **GROUPING SETS** let you specify which groupings of aggregations you need.

You can use CUBE or ROLLUP expressions inside GROUPING SETS expressions. Otherwise, you cannot nest multilevel aggregate expressions.

Grouping Functions

You use one of the following three grouping functions with ROLLUP, CUBE, and GROUPING SETS:

- **GROUP_ID** returns one or more numbers, starting with zero (0), to uniquely identify duplicate sets.
- **GROUPING_ID** produces a unique ID for each grouping combination.

- **GROUPING** identifies for each grouping combination whether a column is a part of this grouping. This function also differentiates NULL values in the data from NULL grouping subtotals.

These functions are typically used with multilevel aggregates.

Pre-Aggregating Data in Projections

Queries that use aggregate functions such as **SUM** and **COUNT** can perform more efficiently when they use projections that already contain the aggregated data. This improved efficiency is especially true for queries on large quantities of data.

For example, a power grid company reads 30 million smart meters that provide data at five-minute intervals. The company records each reading in a database table. Over a given year, three trillion records are added to this table.

The power grid company can analyze these records with queries that include aggregate functions to perform the following tasks:

- Establish usage patterns.
- Detect fraud.
- Measure correlation to external events such as weather patterns or pricing changes.

To optimize query response time, you can create an aggregate projection, which stores the data is stored after it is aggregated.

Aggregate Projections

Vertica provides three types of projections for storing data that is returned from aggregate functions or expressions:

- **Projection that contains expressions:** Projection with columns whose values are calculated from anchor table columns.
- **Live aggregate projection:** Projection that contains columns with values that are aggregated from columns in its anchor table. You can also define live aggregate projections that include **user-defined transform functions**.

- **Top-K projection:** Type of live aggregate projection that returns the top k rows from a partition of selected rows. Create a Top-K projection that satisfies the criteria for a Top-K query.

Recommended Use

- Aggregate projections are most useful for queries against large sets of data.
- For optimal query performance, the size of LAP projections should be a small subset of the anchor table—ideally, between 1 and 10 percent of the anchor table, or smaller, if possible.
- Because you cannot update or delete data in tables that use aggregate projections, use aggregate projections for tables whose data is cumulative and require infrequent updates and deletions.

Requirements

- To use aggregate projections, you must set the appropriate configuration parameters (see [Enabling Live Aggregate Projections and Projections with Expressions](#)).
- In the event of manual recovery from an unclean database shutdown, live aggregate projections might require some time before they are refreshed.

Enabling Live Aggregate Projections and Projections with Expressions

To create live aggregate and Top-K projections, or projections with expressions, the following configuration parameters must be set to 1 (default setting):

Configuration Parameter	Description	Enable	Disable
EnableGroupByProjections	Enables live aggregate projections	1	0
EnableTopKProjections	Enables Top-K projections	1	0
EnableExprsInProjections	Enables projections with expressions.	1	0

To check current settings, use `SHOW CURRENT`. For example:

```
=> SHOW CURRENT EnableGroupByProjections;
 level |          name          | setting
-----+-----+-----
DEFAULT | EnableGroupByProjections | 1
(1 row)
```

Live Aggregate Projections

A live aggregate projection contains columns with values that are aggregated from columns in its anchor table. When you load data into the table, Vertica aggregates the data before loading it into the live aggregate projection. On subsequent loads—for example, through `INSERT` or `COPY`—Vertica recalculates aggregations with the new data and updates the projection.

Functions Supported for Live Aggregate Projections

Vertica can aggregate results in live aggregate projections from the following aggregate functions:

- `SUM`
- `MAX`
- `MIN`
- `COUNT`

Aggregate Functions with `DISTINCT`

Live aggregate projections can support queries that include aggregate functions qualified with the keyword `DISTINCT`. The following requirements apply:

- The aggregated expression must evaluate to a non-constant.
- The projection's `GROUP BY` clause must specify the aggregated expression.

For example, the following query uses `SUM(DISTINCT)` to calculate the total of all unique salaries in a given region:

```
SELECT customer_region, SUM(DISTINCT annual_income)::INT
FROM customer_dimension GROUP BY customer_region;
```

This query can use the following live aggregate projection, which specifies the aggregated column (`annual_income`) in its `GROUP BY` clause:

```
CREATE PROJECTION public.TotalRegionalIncome
(
  customer_region,
  annual_income,
  Count
)
AS
SELECT customer_dimension.customer_region,
       customer_dimension.annual_income,
       count(*) AS Count
FROM public.customer_dimension
GROUP BY customer_dimension.customer_region,
         customer_dimension.annual_income
;
```

Note: This projection includes the aggregate function `COUNT`, which here serves no logical objective; it is included only because live aggregate projections require at least one aggregate function.

Creating Live Aggregate Projections

You define a live aggregate projection with the following syntax:

```
=> CREATE PROJECTION proj-name AS
     SELECT select-expression FROM table
     GROUP BY group-expression;
```

For full syntax options, see [CREATE PROJECTION \(Live Aggregate Projections\)](#).

For example:

```
=> CREATE PROJECTION clicks_agg AS
     SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks
     GROUP BY page_id, click_time::DATE KSAFE 1;
```

For an extended discussion, see [Live Aggregate Projection Example](#).

Requirements

The following requirements apply to live aggregate projections:

- [Live aggregate projections must be enabled](#): configuration parameter `EnableGroupByProjections` must be set to 1 (default).
- The projection cannot be unsegmented.

- [SELECT](#) and [GROUP BY](#) columns must be in the same order. [GROUP BY](#) expressions must be at the beginning of the [SELECT](#) list.

Restrictions

The following restrictions apply to live aggregate projections:

- Vertica does not regard the projections as superprojections, even one that includes all table columns.
- You cannot perform the following operations on anchor table data:
[DELETE](#)
[UPDATE](#)
[MERGE](#)

Note: To modify existing anchor table data, you must first drop all live aggregate and Top-K projections that are associated with it, using [DROP PROJECTION](#).

- The projections can reference only one table.

Live Aggregate Projection Example

This example shows how you can track user clicks on a given web page using the following `clicks` table:

```
=> CREATE TABLE clicks(  
    user_id INTEGER,  
    page_id INTEGER,  
    click_time TIMESTAMP NOT NULL);
```

You can aggregate user-specific activity with the following query:

```
=> SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks  
WHERE click_time::DATE = '2015-04-30'  
GROUP BY page_id, click_time::DATE ORDER BY num_clicks DESC;
```

To facilitate performance of this query, create a live aggregate projection that counts the number of clicks per user:

```
=> CREATE PROJECTION clicks_agg AS  
SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks  
GROUP BY page_id, click_time::DATE KSAFE 1;
```

When you query the `clicks` table on user clicks, Vertica typically directs the query to the live aggregate projection `clicks_agg`. As additional data is loaded into `clicks`, Vertica pre-

aggregates the new data and updates `clicks_agg`, so queries always return with the latest data.

For example:

```
=> SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks
      WHERE click_time::DATE = '2015-04-30' GROUP BY page_id, click_time::DATE
      ORDER BY num_clicks DESC;
 page_id | click_date | num_clicks
-----+-----+-----
      2002 | 2015-04-30 |          10
      3003 | 2015-04-30 |           3
      2003 | 2015-04-30 |           1
      2035 | 2015-04-30 |           1
     12034 | 2015-04-30 |           1
(5 rows)
```

Top-K Projections

A Top-K query returns the top k rows from partitions of selected rows. Top-K projections can significantly improve performance of Top-K queries. For example, you can define a table that stores gas meter readings with three columns: gas meter ID, time of meter reading, and the read value:

```
=> CREATE TABLE readings (
      meter_id INT,
      reading_date TIMESTAMP,
      reading_value FLOAT);
```

Given this table, the following Top-K query returns the five most recent meter readings for a given meter:

```
SELECT meter_id, reading_date, reading_value FROM readings
      LIMIT 5 OVER (PARTITION BY meter_id ORDER BY reading_date DESC);
```

To improve the performance of this query, you can create a Top-K projection, which is a special type of live aggregate projection:

```
=> CREATE PROJECTION readings_topk (meter_id, recent_date, recent_value)
      AS SELECT meter_id, reading_date, reading_value FROM readings
      LIMIT 5 OVER (PARTITION BY meter_id ORDER BY reading_date DESC);
```

After you create this Top-K projection and load its data (through [START_REFRESH](#) or [REFRESH](#)), Vertica typically redirects the query to the projection and returns with the pre-aggregated data.

Creating Top-K Projections

You define a Top-K projection with the following syntax:

```
CREATE PROJECTION proj-name [(proj-column-spec)]  
  AS SELECT select-expression FROM table  
  LIMIT num-rows OVER (PARTITION BY expression ORDER BY column-expr);
```

For full syntax options, see [CREATE PROJECTION \(Live Aggregate Projections\)](#).

For example:

```
=> CREATE PROJECTION readings_topk (meter_id, recent_date, recent_value)  
  AS SELECT meter_id, reading_date, reading_value FROM readings  
  LIMIT 5 OVER (PARTITION BY meter_id ORDER BY reading_date DESC);
```

For an extended discussion, see [Top-K Projection Examples](#).

Requirements

The following requirements apply to Top-K projections:

- [Top-K projections must be enabled](#): configuration parameter `EnableTopKProjections` must be set to 1 (default).
- The projection cannot be unsegmented.
- The [window partition clause](#) must use `PARTITION BY`.
- Columns in `PARTITION BY` and `ORDER BY` clauses must be the first columns specified in the `SELECT` list.
- You must use the `LIMIT` option to create a Top-K projection, instead of subqueries. For example, the following `SELECT` statements are equivalent:

```
=> SELECT symbol, trade_time last_trade, price last_price FROM (  
  SELECT symbol, trade_time, price, ROW_NUMBER()  
  OVER(PARTITION BY symbol ORDER BY trade_time DESC) rn FROM trades) trds WHERE rn <=1;  
  
=> SELECT symbol, trade_time last_trade, price last_price FROM trades  
  LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);
```

Both return the same results:

symbol	last_trade	last_price
AAPL	2011-11-10 10:10:20.5	108.4000
HPQ	2012-10-10 10:10:10.4	42.0500

(2 rows)

A Top-K projection that pre-aggregates data for use by both queries must include the LIMIT option:

```
=> CREATE PROJECTION trades_topk AS
  SELECT symbol, trade_time last_trade, price last_price FROM trades
  LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);
```

Restrictions

The following restrictions apply to Top-K projections:

- Vertica does not regard the projections as superprojections, even one that includes all table columns.
- You cannot perform the following operations on anchor table data:

[DELETE](#)

[UPDATE](#)

[MERGE](#)

Note: To modify existing anchor table data, you must first drop all live aggregate and Top-K projections that are associated with it, using [DROP PROJECTION](#).

- The projections can reference only one table.

Top-K Projection Examples

The following examples show how to query a table with two Top-K projections for the most-recent trade and last trade of the day for each stock symbol.

1. Create a table that contains information about individual stock trades:
 - Stock symbol
 - Timestamp
 - Price per share
 - Number of shares

```
=> CREATE TABLE trades(
  symbol CHAR(16) NOT NULL,
  trade_time TIMESTAMP NOT NULL,
  price NUMERIC(12,4),
  volume INT )
PARTITION BY (EXTRACT(year from trade_time) * 100 +
EXTRACT(month from trade_time));
```

2. Load data into the table:

```
INSERT INTO trades VALUES('AAPL','2010-10-10 10:10:10'::TIMESTAMP,100.00,100);
INSERT INTO trades VALUES('AAPL','2010-10-10 10:10:10.3'::TIMESTAMP,101.00,100);
INSERT INTO trades VALUES ('AAPL','2011-10-10 10:10:10.5'::TIMESTAMP,106.1,1000);
INSERT INTO trades VALUES ('AAPL','2011-10-10 10:10:10.2'::TIMESTAMP,105.2,500);
INSERT INTO trades VALUES ('HPQ','2012-10-10 10:10:10.2'::TIMESTAMP,42.01,400);
INSERT INTO trades VALUES ('HPQ','2012-10-10 10:10:10.3'::TIMESTAMP,42.02,1000);
INSERT INTO trades VALUES ('HPQ','2012-10-10 10:10:10.4'::TIMESTAMP,42.05,100);
COMMIT;
```

3. Create two Top-K projections that obtain the following information from the trades table:

- [Return the most recent trades for each stock symbol.](#)
- [Return the last trade on each trading day.](#)

For each stock symbol, return the most recent trade.

```
=> CREATE PROJECTION trades_topk_a AS SELECT symbol, trade_time last_trade, price last_price
FROM trades LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);

=> SELECT symbol, trade_time last_trade, price last_price FROM trades
LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);
```

symbol	last_trade	last_price
HPQ	2012-10-10 10:10:10.4	42.0500
AAPL	2011-10-10 10:10:10.5	106.1000

(2 rows)

For each stock symbol, return the last trade on each trading day.

```
=> CREATE PROJECTION trades_topk_b
AS SELECT symbol, trade_time::DATE trade_date, trade_time, price close_price, volume
FROM trades LIMIT 1 OVER(PARTITION BY symbol, trade_time::DATE ORDER BY trade_time DESC);

=> SELECT symbol, trade_time::DATE trade_date, trade_time, price close_price, volume
FROM trades LIMIT 1 OVER(PARTITION BY symbol, trade_time::DATE ORDER BY trade_time DESC);
```

symbol	trade_date	trade_time	close_price	volume
--------	------------	------------	-------------	--------

```
HPQ          | 2012-10-10 | 2012-10-10 10:10:10.4 | 42.0500 | 100  
AAPL        | 2011-10-10 | 2011-10-10 10:10:10.5 | 106.1000 | 1000  
AAPL        | 2010-10-10 | 2010-10-10 10:10:10.3 | 101.0000 | 100  
(3 rows)
```

In each scenario, Vertica redirects queries on the `trades` table to the appropriate Top-K projection and returns the aggregated data from them. As additional data is loaded into this table, Vertica pre-aggregates the new data and updates the Top-K projections, so queries always return with the latest data.

Pre-Aggregating UDTF Results

`CREATE PROJECTION` can define live aggregate projections that invoke user-defined transform functions (UDTFs). To minimize overhead when you query those projections, Vertica processes these functions in the background and stores their results on disk.

Important: Currently, live aggregate projections can only reference UDTFs that are developed in C++.

Defining Projections with UDTFs

The projection definition characterizes UDTFs in one of two ways:

- Identifies the UDTF as a *pre-pass UDTF*, which transforms newly loaded data before it is stored in the projection ROS containers.
- Identifies the UDTF as a *batch UDTF*, which aggregates and stores projection data.

The projection definition identifies a UDTF as a pre-pass UDTF or batch UDTF in its [window partition clause](#), through the keywords `PREPASS` or `BATCH`. A projection can specify one pre-pass or batch UDTF or include both (see [UDTF Specification Options](#)).

In all cases, the projection is implicitly segmented and ordered on the `PARTITION BY` columns.

UDTF Specification Options

Projections can invoke batch and pre-pass UDTFs singly or in combination.

Single Pre-Pass UDTF

Vertica invokes the pre-pass UDTF when you load data into the projection's anchor table—for example through COPY or INSERT statements. A pre-pass UDTF transforms the new data and then stores the transformed data in the projection's ROS containers.

Use the following syntax:

```
=> CREATE PROJECTION projection-name AS SELECT ..., udtf(args)
      OVER(PARTITION PREPASS BY partition-col-exprs) FROM table-ref;
```

Single Batch UDTF

When invoked singly, a batch UDTF transforms and aggregates projection data on mergeout, data load, and query operations. The UDTF stores aggregated results in the projection's ROS containers. Aggregation is cumulative across mergeout and load operations, and is completed (if necessary) on query execution.

Use the following syntax:

```
=> CREATE PROJECTION projection-name AS SELECT ..., udtf(args)
      OVER(PARTITION BATCH BY partition-cols) AS (batch-output-columns) FROM table-ref;
```

Combined Pre-Pass and Batch UDTFs

You can define a projection with a subquery that invokes a pre-pass UDTF. The pre-pass UDTF returns transformed data to the outer batch query. The batch UDTF then iteratively aggregates results across mergeout operations. It completes aggregation (if necessary) on query execution.

Use the following syntax:

```
=> CREATE PROJECTION projection-name AS SELECT ..., batch-udtf(batch-args)
      OVER ( PARTITION BATCH BY partition-cols ) AS (batch-output-columns)
      FROM ( SELECT ..., prepass-udtf(prepass-args)
            OVER ( PARTITION PREPASS BY partition-cols) AS (prepass-output-columns)
            FROM table-ref ) sq-ref;
```

Important: The outer batch UDTF arguments *batch-args* must exactly match the output columns returned by the pre-pass UDTF, in name and order.

Examples

Single pre-pass UDTF

The following example shows how to use the UDTF `text_index`, which extracts from a text document strings that occur more than once.

The following projection specifies to invoke `text_index` as a pre-pass UDTF:

```
=> CREATE TABLE documents ( doc_id INT PRIMARY KEY, text VARCHAR(140));

=> CREATE PROJECTION index_proj
  AS SELECT doc_id, text_index(doc_id, text)
  OVER (PARTITION PREPASS BY doc_id) FROM documents;
```

The UDTF is invoked whenever data is loaded into the anchor table `documents`. `text_index` transforms the newly loaded data, and Vertica stores the transformed data in the live aggregate projection ROS containers.

So, if you load the following data into `documents`:

```
=> INSERT INTO documents VALUES (100, 'A SQL Query walks into a bar. In one corner of the bar are two
tables.

                                The Query walks up to the tables and asks - Mind if I join you?');

OUTPUT
-----
      1
(1 row)
```

`text_index` transforms the newly loaded data and stores it in the projection ROS containers. When you query the projection, it returns with the following results:

doc_id	frequency	term
100	2	bar
100	2	Query
100	2	tables
100	2	the
100	2	walks

Combined Pre-Pass and Batch UDTFs

The following projection specifies pre-pass and batch UDTFs `stv_intersect` and `aggregate_classified_points`, respectively:

```
CREATE TABLE points( point_id INTEGER, point_type VARCHAR(10), coordinates GEOMETRY(100));

CREATE PROJECTION aggregated_proj
  AS SELECT point_type, aggregate_classified_points( sq.point_id, sq.polygon_id)
  OVER (PARTITION BATCH BY point_type)
  FROM
    (SELECT point_type, stv_intersect(
```

```
point_id, coordinates USING PARAMETERS index='polygons' )  
OVER (PARTITION PREPASS BY point_type) AS (point_id, polygon_id) FROM points) sq;
```

The pre-pass query UDTF `stv_intersect` returns its results (a set of point and matching polygon IDs) to the outer batch query. The outer batch query then invokes the UDTF `aggregate_classified_points`. Vertica aggregates the result set that is returned by `aggregate_classified_points` whenever a mergeout operation consolidates projection data. Final aggregation (if necessary) occurs when the projection is queried.

The batch UDTF arguments must exactly match the output columns returned by the pre-pass UDTF `stv_intersect`, in name and order. In this example, the pre-pass subquery explicitly names the pre-pass UDTF output columns `point_id` and `polygon_id`. Accordingly, the batch UDTF arguments match them in name and order: `sq.point_id` and `sq.polygon_id`.

Aggregating Data Through Expressions

You can create projections where one or more columns are defined by expressions. An expression can reference one or more anchor table columns. For example, the following table contains two integer columns, `a` and `b`:

```
=> CREATE TABLE values (a INT, b INT);
```

You can create a projection with an expression that calculates the value of column `c` as the product of `a` and `b`:

```
=> CREATE PROJECTION values_product (a, b, c)  
AS SELECT a, b, a*b FROM values SEGMENTED BY HASH(a) ALL NODES KSAFE;
```

When you load data into this projection, Vertica resolves the expression `a*b` in column `c`. You can then query the projection instead of the anchor table. Vertica returns the pre-calculated data and avoids the overhead otherwise incurred by resource-intensive computations.

Using expressions in projections also lets you sort or segment data on the calculated results of an expression instead of sorting on single column values.

Note: If a projection with expressions also includes aggregate functions such as [SUM](#) or [COUNT](#), Vertica treats it like a [live aggregate projection](#).

Support for User-Defined Scalar Functions

Important: Currently, support for pre-aggregating UDSF results is limited to C++.

Vertica treats user-defined scalar functions (UDSFs) like other expressions. On each load operation, the UDSF is invoked and returns its results. Vertica stores these results on disk, and returns them when you query the projection directly.

In the following example, the projection `points_p1` specifies the UDSF `zorder`, which is invoked whenever data is loaded in the anchor table `points`. When data is loaded into the projection, Vertica invokes this function and stores its results for fast access by future queries.

```
=> CREATE TABLE points(point_id INTEGER, lat NUMERIC(12,9), long NUMERIC(12,9));  
  
=> CREATE PROJECTION points_p1  
  AS SELECT point_id, lat, long, zorder(lat, long) zorder FROM points  
  ORDER BY zorder(lat, long) SEGMENTED BY hash(point_id) ALL NODES;
```

Requirements

- [Projections with expressions must be enabled.](#)
- Any `ORDER BY` expression must be in the `SELECT` list.
- All projection columns must be named.

Restrictions

- You can delete and update data in the anchor table, but you cannot perform a merge operation.
- Unlike live aggregate projections, Vertica does not redirect queries with expressions to an equivalent existing projection.
- Projection expressions must be immutable—that is, they must always return the same result. For example, a projection cannot include expressions that use `TO CHAR` (depends on locale) or `RANDOM` (returns different value at each invocation).
- Projection expressions cannot include Vertica meta-functions such as `ADVANCE_EPOCH`, `ANALYZE_STATISTICS`, `EXPORT_TABLES`, or `START_REFRESH`.

Querying Data Through Expressions Example

The following example uses a table that contains two integer columns, `a` and `b`:

```
=> CREATE TABLE values (a INT, b INT);
```

You can create a projection with an expression that calculates the value of column *c* as the product of *a* and *b*:

```
=> CREATE PROJECTION values_product (a, b, c)
  AS SELECT a, b, a*b FROM values SEGMENTED BY HASH(a) ALL NODES KSAFE;
```

To query this projection, use the name that Vertica assigns to it or to its buddy projections. For example, the following queries target different instances of the projection defined earlier, and return the same results:

```
=> SELECT * FROM values_product_b0;
=> SELECT * FROM values_product_b1;
```

The following example queries the anchor table:

```
=> SELECT * FROM values;
 a | b
----+----
 3 | 11
 3 | 55
 8 |  9
 8 | 23
16 | 41
22 | 111
```

Given the projection created earlier, querying that projection returns the following values:

```
VMart=> SELECT * FROM values_product_b0;
 a | b | product
----+----+-----
 3 | 11 |      33
 3 | 55 |     165
 8 |  9 |      72
 8 | 23 |     184
16 | 41 |     656
22 | 111 |    2442
```

System Table Fields

You can query the following system table fields for information about live aggregate projections, Top-K projections, and projections with expressions:

Table	Fields
TABLES	HAS_AGGREGATE_PROJECTION

Table	Fields
PROJECTIONS	AGGREGATE_TYPE
	HAS_EXPRESSIONS
	AGGREGATE_TYPE
PROJECTION_COLUMNS	COLUMN_EXPRESSION
	IS_AGGREGATE
	IS_EXPRESSION
	ORDER_BY_POSITION
	ORDER_BY_TYPE
	PARTITION_BY_POSITION

Event Series Joins

An event series join is a Vertica SQL extension that enables the analysis of two series when their measurement intervals don't align precisely, such as with mismatched timestamps. You can compare values from the two series directly, rather than having to normalize the series to the same measurement interval.

Event series joins are an extension of [Outer Joins](#), but instead of padding the non-preserved side with NULL values when there is no match, the event series join pads the non-preserved side values that it interpolates from the previous value.

The difference in how you write a regular join versus an event series join is the INTERPOLATE predicate, which is used in the ON clause. For example, the following two statements show the differences, which are shown in greater detail in [Writing Event Series Joins](#).

Regular full outer join	Event series join
<pre>SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON (h.time = a.time);</pre>	<pre>SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON (h.time INTERPOLATE PREVIOUS VALUE a.time);</pre>

Similar to regular joins, an event series join has inner and outer join modes, which are described in the topics that follow.

For full syntax, including notes and restrictions, see [INTERPOLATE](#) in the SQL Reference Manual

Sample Schema for Event Series Joins Examples

If you don't plan to run the queries and just want to look at the examples, you can skip this topic and move straight to [Writing Event Series Joins](#).

Schema of hTicks and aTicks Tables

The examples that follow use the following hTicks and aTicks tables schemas:

```
CREATE TABLE hTicks (
  stock VARCHAR(20),
  time TIME,
  price NUMERIC(8,2)
);
CREATE TABLE aTicks (
```

```
stock VARCHAR(20),
time TIME,
price NUMERIC(8,2)
);
```

Although **TIMESTAMP** is more commonly used for the event series column, the examples in this topic use **TIME** to keep the output simple.

```
INSERT INTO hTicks VALUES ('HPQ', '12:00', 50.00);
INSERT INTO hTicks VALUES ('HPQ', '12:01', 51.00);
INSERT INTO hTicks VALUES ('HPQ', '12:05', 51.00);
INSERT INTO hTicks VALUES ('HPQ', '12:06', 52.00);
INSERT INTO aTicks VALUES ('ACME', '12:00', 340.00);
INSERT INTO aTicks VALUES ('ACME', '12:03', 340.10);
INSERT INTO aTicks VALUES ('ACME', '12:05', 340.20);
INSERT INTO aTicks VALUES ('ACME', '12:05', 333.80);
COMMIT;
```

Output of the two tables:

hTicks	aTicks																														
<pre>=> SELECT * FROM hTicks;</pre> <p>There are no entry records between 12:02–12:04:</p> <table border="1"> <thead> <tr> <th>stock</th> <th>time</th> <th>price</th> </tr> </thead> <tbody> <tr> <td>HPQ</td> <td>12:00:00</td> <td>50.00</td> </tr> <tr> <td>HPQ</td> <td>12:01:00</td> <td>51.00</td> </tr> <tr> <td>HPQ</td> <td>12:05:00</td> <td>51.00</td> </tr> <tr> <td>HPQ</td> <td>12:06:00</td> <td>52.00</td> </tr> </tbody> </table> <p>(4 rows)</p>	stock	time	price	HPQ	12:00:00	50.00	HPQ	12:01:00	51.00	HPQ	12:05:00	51.00	HPQ	12:06:00	52.00	<pre>=> SELECT * FROM aTicks;</pre> <p>There are no entry records at 12:01, 12:02 and at 12:04:</p> <table border="1"> <thead> <tr> <th>stock</th> <th>time</th> <th>price</th> </tr> </thead> <tbody> <tr> <td>ACME</td> <td>12:00:00</td> <td>340.00</td> </tr> <tr> <td>ACME</td> <td>12:03:00</td> <td>340.10</td> </tr> <tr> <td>ACME</td> <td>12:05:00</td> <td>340.20</td> </tr> <tr> <td>ACME</td> <td>12:05:00</td> <td>333.80</td> </tr> </tbody> </table> <p>(4 rows)</p>	stock	time	price	ACME	12:00:00	340.00	ACME	12:03:00	340.10	ACME	12:05:00	340.20	ACME	12:05:00	333.80
stock	time	price																													
HPQ	12:00:00	50.00																													
HPQ	12:01:00	51.00																													
HPQ	12:05:00	51.00																													
HPQ	12:06:00	52.00																													
stock	time	price																													
ACME	12:00:00	340.00																													
ACME	12:03:00	340.10																													
ACME	12:05:00	340.20																													
ACME	12:05:00	333.80																													

Example Query Showing Gaps

A full outer join shows the gaps in the timestamps:

```
=> SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON h.time = a.time;
```

stock	time	price	stock	time	price
HPQ	12:00:00	50.00	ACME	12:00:00	340.00
HPQ	12:01:00	51.00			
HPQ	12:05:00	51.00	ACME	12:05:00	333.80
HPQ	12:05:00	51.00	ACME	12:05:00	340.20
HPQ	12:06:00	52.00			
			ACME	12:03:00	340.10

(6 rows)

Schema of Bid and Asks Tables

The examples that follow use the following hTicks and aTicks tables.

```
CREATE TABLE bid(stock VARCHAR(20), time TIME, price NUMERIC(8,2));
CREATE TABLE ask(stock VARCHAR(20), time TIME, price NUMERIC(8,2));
INSERT INTO bid VALUES ('HPQ', '12:00', 100.10);
INSERT INTO bid VALUES ('HPQ', '12:01', 100.00);
INSERT INTO bid VALUES ('ACME', '12:00', 80.00);
INSERT INTO bid VALUES ('ACME', '12:03', 79.80);
INSERT INTO bid VALUES ('ACME', '12:05', 79.90);
INSERT INTO ask VALUES ('HPQ', '12:01', 101.00);
INSERT INTO ask VALUES ('ACME', '12:00', 80.00);
INSERT INTO ask VALUES ('ACME', '12:02', 75.00);
COMMIT;
```

Output of the two tables:

bid	ask																														
<pre>=> SELECT * FROM bid;</pre> <p>There are no entry records for stocks HPQ and ACME at 12:02 and at 12:04:</p> <table border="1"> <thead> <tr> <th>stock</th> <th>time</th> <th>price</th> </tr> </thead> <tbody> <tr><td>HPQ</td><td>12:00:00</td><td>100.10</td></tr> <tr><td>HPQ</td><td>12:01:00</td><td>100.00</td></tr> <tr><td>ACME</td><td>12:00:00</td><td>80.00</td></tr> <tr><td>ACME</td><td>12:03:00</td><td>79.80</td></tr> <tr><td>ACME</td><td>12:05:00</td><td>79.90</td></tr> </tbody> </table> <p>(5 rows)</p>	stock	time	price	HPQ	12:00:00	100.10	HPQ	12:01:00	100.00	ACME	12:00:00	80.00	ACME	12:03:00	79.80	ACME	12:05:00	79.90	<pre>=> SELECT * FROM ask;</pre> <p>There are no entry records for stock HPQ at 12:00 and none for ACME at 12:01:</p> <table border="1"> <thead> <tr> <th>stock</th> <th>time</th> <th>price</th> </tr> </thead> <tbody> <tr><td>HPQ</td><td>12:01:00</td><td>101.00</td></tr> <tr><td>ACME</td><td>12:00:00</td><td>80.00</td></tr> <tr><td>ACME</td><td>12:02:00</td><td>75.00</td></tr> </tbody> </table> <p>(3 rows)</p>	stock	time	price	HPQ	12:01:00	101.00	ACME	12:00:00	80.00	ACME	12:02:00	75.00
stock	time	price																													
HPQ	12:00:00	100.10																													
HPQ	12:01:00	100.00																													
ACME	12:00:00	80.00																													
ACME	12:03:00	79.80																													
ACME	12:05:00	79.90																													
stock	time	price																													
HPQ	12:01:00	101.00																													
ACME	12:00:00	80.00																													
ACME	12:02:00	75.00																													

Example Query Showing Gaps

A full outer join shows the gaps in the timestamps:

```
=> SELECT * FROM bid b FULL OUTER JOIN ask a ON b.time = a.time;
```

stock	time	price	stock	time	price
HPQ	12:00:00	100.10	ACME	12:00:00	80.00
HPQ	12:01:00	100.00	HPQ	12:01:00	101.00
ACME	12:00:00	80.00	ACME	12:00:00	80.00
ACME	12:03:00	79.80			
ACME	12:05:00	79.90			
			ACME	12:02:00	75.00

(6 rows)

Writing Event Series Joins

The examples in this topic contains mismatches between timestamps—just as you'd find in real life situations; for example, there could be a period of inactivity on stocks where no trade occurs, which can present challenges when you want to compare two stocks whose timestamps don't match.

The hTicks and aTicks Tables

As described in the [example ticks schema](#), tables, hTicks is missing input rows for 12:02, 12:03, and 12:04, and aTicks is missing inputs at 12:01, 12:02, and 12:04.

hTicks	aTicks
<pre>=> SELECT * FROM hTicks; stock time price -----+-----+----- HPQ 12:00:00 50.00 HPQ 12:01:00 51.00 HPQ 12:05:00 51.00 HPQ 12:06:00 52.00 (4 rows)</pre>	<pre>=> SELECT * FROM aTicks; stock time price -----+-----+----- ACME 12:00:00 340.00 ACME 12:03:00 340.10 ACME 12:05:00 340.20 ACME 12:05:00 333.80 (4 rows)</pre>

Querying Event Series Data with Full Outer Joins

Using a traditional full outer join, this query finds a match between tables hTicks and aTicks at 12:00 and 12:05 and pads the missing data points with NULL values.

```
=> SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON (h.time = a.time);
stock | time | price | stock | time | price
-----+-----+-----+-----+-----+-----
HPQ   | 12:00:00 | 50.00 | ACME  | 12:00:00 | 340.00
HPQ   | 12:01:00 | 51.00 |      |      |
HPQ   | 12:05:00 | 51.00 | ACME  | 12:05:00 | 333.80
HPQ   | 12:05:00 | 51.00 | ACME  | 12:05:00 | 340.20
HPQ   | 12:06:00 | 52.00 |      |      |
      |      |      | ACME  | 12:03:00 | 340.10
(6 rows)
```

To replace the gaps with interpolated values for those missing data points, use the [INTERPOLATE](#) predicate to create an event series join. The join condition is restricted to the ON clause, which evaluates the equality predicate on the timestamp columns from the two input

tables. In other words, for each row in outer table hTicks, the ON clause predicates are evaluated for each combination of each row in the inner table aTicks.

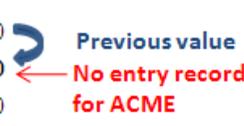
Simply rewrite the full outer join query to use the INTERPOLATE predicate with the required PREVIOUS VALUE keywords. Note that a full outer join on event series data is the most common scenario for event series data, where you keep all rows from both tables

```
=> SELECT * FROM hTicks h FULL OUTER JOIN aTicks a
    ON (h.time INTERPOLATE PREVIOUS VALUE a.time);
```

Vertica interpolates the missing values (which appear as NULL in the full outer join) using that table's previous value:

stock	time	price	stock	time	price
HPQ	12:00:00	50.00	ACME	12:00:00	340.00
HPQ	12:01:00	51.00	ACME	12:00:00	340.00
HPQ	12:01:00	51.00	ACME	12:03:00	340.10
HPQ	12:05:00	51.00	ACME	12:05:00	333.80
HPQ	12:05:00	51.00	ACME	12:05:00	340.20
HPQ	12:06:00	52.00	ACME	12:05:00	340.20

(6 rows)

 A blue arrow points from the ACME price of 340.00 at 12:00:00 to the ACME price of 340.00 at 12:01:00. A red arrow points from the text 'No entry record for ACME' to the ACME price of 340.00 at 12:01:00.

Note: The output ordering above is different from the regular full outer join because in the event series join, interpolation occurs independently for each stock (hTicks and aTicks), where the data is partitioned and sorted based on the equality predicate. This means that interpolation occurs within, not across, partitions.

If you review the regular full outer join output, you can see that both tables have a match in the time column at 12:00 and 12:05, but at 12:01, there is no entry record for ACME. So the operation interpolates a value for ACME (ACME, 12:00, 340) based on the previous value in the aTicks table.

Querying Event Series Data with Left Outer Joins

You can also use left and right outer joins. You might, for example, decide you want to preserve only hTicks values. So you'd write a left outer join:

```
=> SELECT * FROM hTicks h LEFT OUTER JOIN aTicks a
    ON (h.time INTERPOLATE PREVIOUS VALUE a.time);
stock | time   | price | stock | time   | price
-----+-----+-----+-----+-----+-----
HPQ   | 12:00:00 | 50.00 | ACME  | 12:00:00 | 340.00
```

```
HPQ | 12:01:00 | 51.00 | ACME | 12:00:00 | 340.00  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 333.80  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 340.20  
HPQ | 12:06:00 | 52.00 | ACME | 12:05:00 | 340.20  
(5 rows)
```

Here's what the same data looks like using a traditional left outer join:

```
=> SELECT * FROM hTicks h LEFT OUTER JOIN aTicks a ON h.time = a.time;  
stock | time | price | stock | time | price  
-----+-----+-----+-----+-----+-----  
HPQ | 12:00:00 | 50.00 | ACME | 12:00:00 | 340.00  
HPQ | 12:01:00 | 51.00 | | | |  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 333.80  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 340.20  
HPQ | 12:06:00 | 52.00 | | | |  
(5 rows)
```

Note that a right outer join has the same behavior with the preserved table reversed.

Querying Event Series Data with Inner Joins

Note that INNER event series joins behave the same way as normal ANSI SQL-99 joins, where all gaps are omitted. Thus, there is nothing to interpolate, and the following two queries are equivalent and return the same result set:

A regular inner join:

```
=> SELECT * FROM hTicks h JOIN aTicks a  
ON (h.time INTERPOLATE PREVIOUS VALUE a.time);  
stock | time | price | stock | time | price  
-----+-----+-----+-----+-----+-----  
HPQ | 12:00:00 | 50.00 | ACME | 12:00:00 | 340.00  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 333.80  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 340.20  
(3 rows)
```

An event series inner join:

```
=> SELECT * FROM hTicks h INNER JOIN aTicks a ON (h.time = a.time);  
stock | time | price | stock | time | price  
-----+-----+-----+-----+-----+-----  
HPQ | 12:00:00 | 50.00 | ACME | 12:00:00 | 340.00  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 333.80  
HPQ | 12:05:00 | 51.00 | ACME | 12:05:00 | 340.20  
(3 rows)
```

The Bid and Ask Tables

Using the [example schema](#) for the `bid` and `ask` tables, write a full outer join to interpolate the missing data points:

```
=> SELECT * FROM bid b FULL OUTER JOIN ask a  
      ON (b.stock = a.stock AND b.time INTERPOLATE PREVIOUS VALUE a.time);
```

In the below output, the first row for stock HPQ shows nulls because there is no entry record for HPQ before 12:01.

stock	time	price	stock	time	price
ACME	12:00:00	80.00	ACME	12:00:00	80.00
ACME	12:00:00	80.00	ACME	12:02:00	75.00
ACME	12:03:00	79.80	ACME	12:02:00	75.00
ACME	12:05:00	79.90	ACME	12:02:00	75.00
HPQ	12:00:00	100.10			
HPQ	12:01:00	100.00	HPQ	12:01:00	101.00

(6 rows)

Note also that the same row (ACME, 12:02, 75) from the `ask` table appears three times. The first appearance is because no matching rows are present in the `bid` table for the row in `ask`, so Vertica interpolates the missing value using the ACME value at 12:02 (75.00). The second appearance occurs because the row in `bid` (ACME, 12:05, 79.9) has no matches in `ask`. The row from `ask` that contains (ACME, 12:02, 75) is the closest row; thus, it is used to interpolate the values.

If you write a regular full outer join, you can see where the mismatched timestamps occur:

```
=> SELECT * FROM bid b FULL OUTER JOIN ask a ON (b.time = a.time);
```

stock	time	price	stock	time	price
ACME	12:00:00	80.00	ACME	12:00:00	80.00
ACME	12:03:00	79.80			
ACME	12:05:00	79.90			
HPQ	12:00:00	100.10	ACME	12:00:00	80.00
HPQ	12:01:00	100.00	HPQ	12:01:00	101.00
			ACME	12:02:00	75.00

(6 rows)

Event Series Pattern Matching

The SQL [MATCH Clause](#) syntax (described in the SQL Reference Manual) lets you screen large amounts of historical data in search of event patterns. You specify a pattern as a regular expression and can then search for the pattern within a sequence of input events. MATCH provides subclauses for analytic data partitioning and ordering, and the pattern matching occurs on a contiguous set of rows.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- landed on the company home page
- navigated to the product page
- ran a search
- clicked a link from the search results
- made a purchase

Clickstream Funnel Schema

The examples in this topic use this clickstream funnel and the following `clickstream_log` table schema:

```
=> CREATE TABLE clickstream_log (  
  uid INT,           --user ID  
  sid INT,           --browsing session ID, produced by previous sessionization computation  
  ts TIME,           --timestamp that occurred during the user's page visit  
  refURL VARCHAR(20), --URL of the page referencing PageURL  
  pageURL VARCHAR(20), --URL of the page being visited  
  action CHAR(1)     --action the user took after visiting the page ('P' = Purchase, 'V' = View)  
);  
  
INSERT INTO clickstream_log VALUES (1,100,'12:00','website1.com','website2.com/home', 'V');  
INSERT INTO clickstream_log VALUES (1,100,'12:01','website2.com/home','website2.com/floby', 'V');  
INSERT INTO clickstream_log VALUES (1,100,'12:02','website2.com/floby','website2.com/shamwow', 'V');  
INSERT INTO clickstream_log values (1,100,'12:03','website2.com/shamwow','website2.com/buy', 'P');
```

```
INSERT INTO clickstream_log values (2,100,'12:10','website1.com','website2.com/home', 'V');
INSERT INTO clickstream_log values (2,100,'12:11','website2.com/home','website2.com/forks', 'V');
INSERT INTO clickstream_log values (2,100,'12:13','website2.com/forks','website2.com/buy', 'P');
COMMIT;
```

Here's the clickstream_log table's output:

```
=> SELECT * FROM clickstream_log;
uid | sid | ts | refURL | pageURL | action
-----+-----+-----+-----+-----+-----
 1 | 100 | 12:00:00 | website1.com | website2.com/home | V
 1 | 100 | 12:01:00 | website2.com/home | website2.com/floby | V
 1 | 100 | 12:02:00 | website2.com/floby | website2.com/shamwow | V
 1 | 100 | 12:03:00 | website2.com/shamwow | website2.com/buy | P
 2 | 100 | 12:10:00 | website1.com | website2.com/home | V
 2 | 100 | 12:11:00 | website2.com/home | website2.com/forks | V
 2 | 100 | 12:13:00 | website2.com/forks | website2.com/buy | P
(7 rows)
```

Example

This example includes the Vertica [Pattern Matching Functions](#) to analyze users' browsing history over website2.com. It identifies patterns where the user performed the following tasks:

- Landed on website2.com from another web site (Entry)
- Browsed to any number of other pages (Onsite)
- Made a purchase (Purchase)

In the following statement, pattern P (Entry Onsite* Purchase) consist of three event types: Entry, Onsite, and Purchase. When Vertica finds a match in the input table, the associated pattern instance must be an event of type Entry followed by 0 or more events of type Onsite, and an event of type Purchase

```
=> SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       event_name(),
       pattern_id(),
       match_id()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
DEFINE
Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
```

```

Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
PATTERN
P AS (Entry Onsite* Purchase)
ROWS MATCH FIRST EVENT);

```

In the output below, the first four rows represent the pattern for user 1's browsing activity, while the following three rows show user 2's browsing habits.

uid id	sid match_id	ts	refurl	pageurl	action	event_name	pattern_
1	100	12:00:00	website1.com	website2.com/home	V	Entry	
1	1						
1	100	12:01:00	website2.com/home	website2.com/floby	V	Onsite	
1	2						
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	Onsite	
1	3						
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	Purchase	
1	4						
2	100	12:10:00	website1.com	website2.com/home	V	Entry	
1	1						
2	100	12:11:00	website2.com/home	website2.com/forks	V	Onsite	
1	2						
2	100	12:13:00	website2.com/forks	website2.com/buy	P	Purchase	
1	3						

(7 rows)

See Also

- [MATCH Clause](#)
- [Pattern Matching Functions](#)

Using Flex Tables

This guide describes how to use flexible (flex) tables, which are a different kind of database table designed for loading and querying unstructured data, also called *semi-structured* data in your Vertica Analytics Platform. Flex tables can contain only unstructured, raw data, or both unstructured and columnar data. You can create a flex table with or without a schema or real columns. Hybrid tables consist of both unstructured and real columns. Both flex and hybrid tables are fully supported Vertica Analytics Platform tables, stored as projections and with the same K-safety as your database.

Audience

This guide is intended for use by any user or database designer or application developer interested in working with flexible tables in the database.

Prerequisites

This guide assumes that Vertica Analytics Platform Version 8.1.x is installed and running in your environment.

It also assumes that you are familiar with using the Vertica Analytics Platform, especially the following features and commands:

- Loading data with the [COPY](#) statement and its basic parameters
- Using statements to create tables [CREATE TABLE](#) or CTAS (create table as...)
- Altering table definitions with the [ALTER TABLE](#) statement
- Creating and using Views ([CREATE VIEW](#))
- Querying data using the [SELECT](#) statement
- Using functions for your database

If you are not familiar with these tasks and features, see [Getting Started](#).

Getting Started

Getting Started describes the basics of creating, exploring, and using flex tables. The rest of this guide presents *beyond the basics* details using simple examples.

Create a Simple JSON File

Use this JSON data for the exercises in the rest of this section:

```
{"name": "Everest", "type": "mountain", "height": 29029, "hike_safety": 34.1}
{"name": "Mt St Helens", "type": "volcano", "height": 29029, "hike_safety": 15.4}
{"name": "Denali", "type": "mountain", "height": 17000, "hike_safety": 12.2}
{"name": "Kilimanjaro", "type": "mountain", "height": 14000 }
{"name": "Mt Washington", "type": "mountain", "hike_safety": 50.6}
```

1. Copy and paste the JSON data into your favorite editor.
2. Save the file in any convenient location for loading into your Vertica database.

Create a Flex Table and Load Data

1. Create a flex table called mountains:

```
=> CREATE flex table mountains();
```

2. Load the JSON file you saved, using the flex table parser `fjsonparser`:

```
=> COPY mountains from '/home/dbadmin/data/flex/mountains.json'
parser fjsonparser();
  Rows Loaded
-----
                5
(1 row)
```

3. Query values from the sample file:

```
=> SELECT name, type, height from mountains;
  name | type | height
-----+-----+-----
 Everest | mountain | 29029
```

```
Mt St Helens | volcano | 29029
Denali      | mountain | 17000
Kilimanjaro | mountain | 14000
Mt Washington | mountain |
(5 rows)
```

You have now created a flex table and loaded data. Next, learn more about using flex table data in your database.

Query More of Your Flex Table

1. Query your flex table to see the data you loaded as it is stored in the `__raw__` column. The example illustrates the table contents, with Return characters added for illustration:

```
=> \x
Expanded display is on.
=> SELECT * from mountains;
[ RECORD 1 ]+-----
__identity__ | 1
__raw__      | \001\000\000\000,\000\000\000\004\000\000\000\024\000\000\000\031\000\000\000\
035\000\000\000$\000\000\0002902934.1Everestmountain\004\000\000\000\024\000\000\000\032\000\
000\000%\000\000\000)\000\000\000heighthike_safetynametype
[ RECORD 2 ]+-----
__identity__ | 2
__raw__      | \001\000\000\0000\000\000\000\004\000\000\000\024\000\000\000\031\000\000\000\
035\000\000\000)\000\000\0002902915.4Mt St
Helensvolcano\004\000\000\000\024\000\000\000\032\000\
000\000%\000\000\000)\000\000\000heighthike_safetynametype
[ RECORD 3 ]+-----
__identity__ | 3
__raw__      | \001\000\000\000+\000\000\000\004\000\000\000\024\000\000\000\031\000\000\000\
035\000\000\000#\000\000\0001700012.2Denalimountain\004\000\000\000\024\000\000\000\032\000\000\
\000%\000\000\000)\000\000\000heighthike_safetynametype
[ RECORD 4 ]+-----
__identity__ | 4
__raw__      | \001\000\000\000(\000\000\000\003\000\000\000\020\000\000\000\025\000\000\000\
000\000\00014000Kilimanjaromountain\003\000\000\000\020\000\000\000\026\000\000\000\032\000\
000\000heightnametype
[ RECORD 5 ]+-----
__identity__ | 5
__raw__      | \001\000\000\000)\000\000\000\003\000\000\000\020\000\000\000\024\000\000\000\
000\000\00050.6Mt Washingtonmountain\003\000\000\000\020\000\000\000\033\000\000\000\037\000\
000\000hike_safetynametype
```

2. Use the `mapToString()` function (with the `__raw__` column of `mountains`) to inspect its contents in readable JSON text format:

```
=> SELECT maptostring(__raw__) from mountains;
                                MAPTOSTRING
-----
{
  "height" : "29029",
  "hike_safety" : "34.1",
  "name" : "Everest",
  "type" : "mountain"
}
{
  "height" : "29029",
  "hike_safety" : "15.4",
  "name" : "Mt St Helens",
  "type" : "volcano"
}
{
  "height" : "17000",
  "hike_safety" : "12.2",
  "name" : "Denali",
  "type" : "mountain"
}
{
  "height" : "14000",
  "name" : "Kilimanjaro",
  "type" : "mountain"
}
{
  "hike_safety" : "50.6",
  "name" : "Mt Washington",
  "type" : "mountain"
}
```

3. Now, use the `compute_flexible_keys()` function to populate the `mountain_keys` table. Vertica generates this table automatically when you create your flex table.

```
=> SELECT compute_flexible_keys('mountains');
       compute_flexible_keys
-----
Please see public.mountains_keys for updated keys
(1 row)
```

4. Query the keys table (`mountains_keys`), and examine the results:

```
=> SELECT * from public.mountains_keys;
 key_name | frequency | data_type_guess
-----+-----+-----
hike_safety |          4 | varchar(20)
name       |          5 | varchar(26)
height    |          4 | varchar(20)
```

```
type          |          5 | varchar(20)
(4 rows)
```

Build a Flex Table View

1. Use the `build_flextable_view()` function to populate a view generated from the `mountains_keys` table.

```
=> SELECT build_flextable_view('mountains');
          build_flextable_view
-----
The view public.mountains_view is ready for querying
(1 row)
```

2. Query the view `mountains_view`:

```
=> SELECT * from public.mountains_view;
hike_safety | name          | type   | height
-----+-----+-----+-----
50.6       | Mt Washington | mountain |
34.1       | Everest      | mountain | 29029
22.8       | Kilimanjaro  | mountain | 14000
15.4       | Mt St Helens | volcano  | 29029
12.2       | Denali       | mountain | 17000
(5 rows)
```

3. Use the `view_columns` system table to query the `column_name` and `data_type` columns for `mountains_view`:

```
=> SELECT column_name, data_type from view_columns where table_name = 'mountains_view';
column_name | data_type
-----+-----
hike_safety | varchar(20)
name        | varchar(26)
type        | varchar(20)
height      | varchar(20)
(4 rows)
```

4. Review the query results:

- Notice the `data_type` column, its values and sizes. These are calculated when you compute keys for your flex table with `compute_flextable_keys()`.

- Did you also notice the `data_type_guess` column when you queried the `mountains_keys` table after invoking that function?
5. With the `data_type` information from `mountains_view`, override the `data_type_guess` for `hike_safety`. Then, `COMMIT` the change, and rebuild the view with `build_flexitable_view()`:

```
=> UPDATE mountains_keys SET data_type_guess = 'float' where key_name = 'hike_safety';
OUTPUT
-----
      1
(1 row)

=> commit;
=> SELECT build_flexitable_view('mountains');
           build_flexitable_view
-----
The view public.mountains_view is ready for querying
(1 row)
```

6. Next, use the `view_columns` system table. Notice that `hike_safety` is now a float data type:

```
=> SELECT column_name, data_type from view_columns where table_name = 'mountains_view';
column_name | data_type
-----+-----
hike_safety | float
name        | varchar(26)
type        | varchar(20)
height      | varchar(20)
(4 rows)
```

Create a Hybrid Flex Table

If you already know that some of the data you load and query regularly needs full Vertica performance and support, you can create a *hybrid* flex table. A hybrid flex table has one or more real columns that you define, and a `__raw__` column to store any unstructured data you load. Querying real columns is faster than querying flexible data in the `__raw__` column. You can define default values for the columns.

1. Create a hybrid flex table, and load the same sample JSON file:

```
=> CREATE flex table mountains_hybrid(name varchar(41) default name::varchar(41), hike_safety
float
```

```
default hike_safety::float);
=> COPY mountains_hybrid from '/home/dbadmin/Downloads/mountains.json' parser fjsonparser();
  Rows Loaded
-----
                5
(1 row)
```

2. Use the `compute_flexable_keys_and_build_view()` function to populate the keys table and build the view for `mountains_hybrid`:

```
=> SELECT compute_flexable_keys_and_build_view('mountains_hybrid');
           compute_flexable_keys_and_build_view
-----
Please see public.mountains_hybrid_keys for updated keys
The view public.mountains_hybrid_view is ready for querying
(1 row)
```

3. Query the `mountains_hybrid_keys` table. Review the `data_type_guesses` column values again. The types list the column definitions you declared when you created the hybrid table:

```
=> SELECT * from mountains_hybrid_keys;
  key_name | frequency | data_type_guess
-----+-----+-----
height    |          4 | varchar(20)
name      |          5 | varchar(41)
type      |          5 | varchar(20)
hike_safety |          4 | float
(4 rows)
```

If you create a basic flex table, and later find you want to promote one or more virtual columns to real columns, see [Materializing Flex Tables](#) to add columns.

Materialize Virtual Columns in a Hybrid Flex Table

After you explore your flex table data, you can promote one or more virtual columns in your flex table to real columns. You do not need to create a separate columnar table.

1. Invoke the `materialize_flextable_columns()` function on the hybrid table, specifying the number of virtual columns to materialize:

```
=> SELECT materialize_flextable_columns('mountains_hybrid', 3);
           materialize_flextable_columns
-----
The following columns were added to the table public.mountains_hybrid:
  type
For more details, run the following query:
SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE
table_schema = 'public' and table_name = 'mountains_hybrid';

(1 row)
```

2. You specified three (3) columns to materialize, but the table was created with two real columns (`name` and `hike_safety`). To fulfill your three-column specification, the function promotes only one other column, `type`. The next example has expanded display to list the columns vertically. Notice the `ADDED` status for the column that was just materialized, rather than `EXISTS` for the two columns you defined when creating the table:

```
=> \x
Expanded display is on.
=> SELECT * from materialize_flextable_columns_results where table_name = 'mountains_hybrid';
-[ RECORD 1 ]-----
table_id      | 45035996273766044
table_schema  | public
table_name    | mountains_hybrid
creation_time | 2013-11-30 20:09:37.765257-05
key_name      | type
status        | ADDED
message       | Added successfully
-[ RECORD 2 ]-----
table_id      | 45035996273766044
table_schema  | public
table_name    | mountains_hybrid
creation_time | 2013-11-30 20:09:37.765284-05
key_name      | hike_safety
status        | EXISTS
message       | Column of same name already exists in table definition
-[ RECORD 3 ]-----
table_id      | 45035996273766044
table_schema  | public
table_name    | mountains_hybrid
creation_time | 2013-11-30 20:09:37.765296-05
key_name      | name
status        | EXISTS
message       | Column of same name already exists in table definition
```

3. Now, display the hybrid table definition, listing the `__raw__` column and the three materialized columns. Flex table data types are derived from the associated keys tables, so you can update them as necessary. Notice that the `__raw__` column has a default NOT

NULL constraint:

```
=> \d mountains_hybrid
List of Fields by Tables
-[ RECORD 1 ]-----
Schema      | public
Table       | mountains_hybrid
Column      | __raw__
Type        | long varbinary(130000)
Size        | 130000
Default     |
Not Null    | t
Primary Key | f
Foreign Key |
-[ RECORD 2 ]-----
Schema      | public
Table       | mountains_hybrid
Column      | name
Type        | varchar(41)
Size        | 41
Default     | (MapLookup(mountains_hybrid.__raw__, 'name'))::varchar(41)
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 3 ]-----
Schema      | public
Table       | mountains_hybrid
Column      | hike_safety
Type        | float
Size        | 8
Default     | (MapLookup(mountains_hybrid.__raw__, 'hike_safety'))::float
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 4 ]-----
Schema      | public
Table       | mountains_hybrid
Column      | type
Type        | varchar(20)
Size        | 20
Default     | (MapLookup(mountains_hybrid.__raw__, 'type'))::varchar(20)
Not Null    | f
Primary Key | f
Foreign Key |
```

You have now completed getting started with flex table basics, hybrid flex tables, and using flex functions.

Understanding Flex Tables

You can create flex tables and then manage them with their associated helper, data, and map functions. Flex tables:

- Do not require schema definitions
- Do not need column definitions
- Have full Unicode support
- Support SQL queries

You can use flex tables to promote data directly from exploration to analytic operations. Flex table features include:

- Ability to load different formats into one flex table, which lets you handle changing structure over time
- Full support of delimited and JSON data
- Extensive SQL queries and built-in analytics for the data you load
- Usability functions, which let you explore your unstructured data and then use built-in functions to materialize the data

Exploration to Promotion

After you create a flex table, you can quickly load data, including social media content in JSON, log files, delimited files, and other information. Previously, working with such data required significant schema design and preparation. Now, you can load and query flex tables in a few steps.

Creating flex tables is similar to creating other tables, except column definitions are optional. When you create flex tables, with or without column definitions, Vertica implicitly adds a real column to your table, called `__raw__`. This column stores loaded data. The `__raw__` column is a `LONG VARBINARY` column with a `NOT NULL` constraint. It contains the documented limits for its data type (see [Long Data Types](#) in the SQL Reference Manual. The `__raw__` column's default maximum width is 130,000 bytes (with an absolute maximum of 32,000,000 bytes). You can change the default width with the `FlexTablesRawSize` configuration parameter.

If you create a flex table without other column definitions, the table includes a second default column, `__identity__`, declared as an auto-incrementing IDENTITY (1,1) column. When no other columns are defined, flex tables use the `__identity__` column for segmentation and sort order.

Loading data into a flex table encodes the record into a VMap type and populates the `__raw__` column. The VMap is a standard dictionary type, pairing keys with string values as virtual columns.

Flex Table Terms

This guide uses the following terms when describing how you work with flex tables to explore and analyze flexible data:

- VMap: An internal map data format.
- Virtual Columns: Key-value pairs contained in a flex table `__raw__` column.
- Real Columns: Fully featured columns in flex or columnar tables.
- Promoted Columns : Virtual columns that have been materialized to real columns.
- Map Keys: Map keys are the virtual column names within VMap data.

Is There Structure in a Flex Table?

The term *unstructured data* (sometimes called *semi-structured* or *Dark Data*) does not indicate that the data you load into flex tables is entirely without structure. However, you may not know the data's composition or the inconsistencies of its design. In some cases, the data may not be relational.

Your data may have some structure (like JSON and delimited data). Data may be semi-structured or stringently structured, but in ways that you either do not know about or do not expect. In this guide, the term *flexible data* encompasses these and other kinds of data. You can load your flexible data directly into a flex table, and query its contents with your favorite SQL SELECT or other statements.

To summarize, you can load data first, without knowing its structure, and then query its content after a few simple transformations. In some cases, you already know the data structure, such as some tweet map keys, like `user.lang`, `user.screen_name`, and `user.url`. If so, you can query these values explicitly as soon as you load the data.

Storing Flex Table Data

While you can store unstructured data in a flex table `__raw__` column, that column is implemented as a real column.

Vertica compresses `__raw__` column data by about one half (1/2). While this factor is less than the compression rate for real columns, the reduction is significant for large amounts (more than 1TB) of unstructured data. After compression is complete, Vertica writes the data to disk (ROS). This approach maintains K-safety in your cluster and supports standard recovery processes should node failures occur. Flex tables are included in full backups (or, if you choose, in object-level backups).

What Happens When You Create Flex Tables?

Whenever you execute a `CREATE FLEX TABLE` statement, Vertica creates three objects, as follows:

- The flexible table (*flex_table*)
- An associated keys table (*flex_table_keys*)
- A default view for the main table (*flex_table_view*)

The `_keys` and `_view` objects are dependents of the parent, *flex_table*. Dropping the flex table also removes its dependents, although you can drop the `_keys` or `_view` objects independently.

You can create a flex table without specifying any column definitions (such as `darkdata`, in the next example). When you do so, Vertica automatically creates two tables, the named flex table (such as `darkdata`) and its associated keys table, `darkdata_keys`:

```
=> CREATE flex table darkdata();
CREATE TABLE
=> \dt dark*
      List of tables
 Schema |      Name      | Kind | Owner  | Comment
-----+-----+-----+-----+-----
 public | darkdata       | table | dbadmin |
 public | darkdata_keys  | table | dbadmin |
(2 rows)
```

Each flex table has two default columns, `__raw__` and `__identity__`. The `__raw__` column exists in every flex table to hold the data you load. The `__identity__` column is auto-incrementing. Vertica uses the `__identity__` column for segmentation and sort order when no other column definitions exist. The flex keys table (`darkdata_keys`) has three columns, as shown:

```
=> SELECT * FROM darkdata;
__identity__ | __raw__
-----+-----
(0 rows)

=> SELECT * FROM darkdata_keys;
key_name | frequency | data_type_guess
-----+-----
(0 rows)
```

Creating a flex table with column definitions (such as `darkdata1`, in the next example) automatically generates a table with the `__raw__` column. However, the table has no `__identity__` column because columns are specified for segmentation and sort order. Two tables are created automatically, as shown in the following example:

```
=> CREATE FLEX TABLE darkdata1 (name VARCHAR);
CREATE TABLE

=> SELECT * FROM darkdata1;
__raw__ | name
-----+-----
(0 rows)

=> \d darkdata1*

                                List of Fields by Tables
 Schema | Table  | Column |          Type          | Size | Default | Not Null | Primary Key | Foreign Key
-----+-----+-----+-----+-----+-----+-----+-----+-----
 public | darkdata1 | __raw__ | long varbinary(130000) | 130000 |         | t        | f           |
 public | darkdata1 | name    | varchar(80)           | 80    |         | f        | f           |
(2 rows)

=> \dt darkdata1*

                                List of tables
 Schema | Name          | Kind | Owner  | Comment
-----+-----+-----+-----+-----
 public | darkdata1     | table | dbadmin |
 public | darkdata1_keys | table | dbadmin |
(2 rows)
```

Creating a flex table with at least one column definition (`darkdata1` in the next example) also generates a table with the `__raw__` column, but not an `__identity__` column. Instead, the specified columns are used for segmentation and sort order. Two tables are also created automatically, as shown in the following example:

```
=> CREATE FLEX TABLE darkdata1 (name VARCHAR);
CREATE TABLE

=> \d darkdata1*

                          List of Fields by Tables
 Schema | Table | Column | Type | Size | Default | Not Null | Primary Key | Foreign Key
-----+-----+-----+-----+-----+-----+-----+-----+-----
 public | darkdata1 | __raw__ | long varbinary(130000) | 130000 | | t | f |
 public | darkdata1 | name | varchar(80) | 80 | | f | f |
(2 rows)

=> \dt darkdata1*

                          List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | darkdata1 | table | dbadmin |
 public | darkdata1_keys | table | dbadmin |
(2 rows)
```

For more examples, see [Creating Flex Tables](#).

Creating Superprojections Automatically

In addition to creating two tables for each flex table, Vertica creates superprojections for both the main flex table and its associated keys table. Using the `\dj` command, you can display the projections created automatically for the `darkdata` and `darkdata1` tables in this set of examples:

```
=> \dj darkdata*

                          List of projections
 Schema | Name | Owner | Node | Comment
-----+-----+-----+-----+-----
 public | darkdata1_b0 | dbadmin | |
 public | darkdata1_b1 | dbadmin | |
 public | darkdata1_keys_super | dbadmin | v_vmart_node0001 |
 public | darkdata1_keys_super | dbadmin | v_vmart_node0003 |
 public | darkdata1_keys_super | dbadmin | v_vmart_node0004 |
 public | darkdata_b0 | dbadmin | |
 public | darkdata_b1 | dbadmin | |
 public | darkdata_keys__super | dbadmin | v_vmart_node0001 |
 public | darkdata_keys_super | dbadmin | v_vmart_node0003 |
 public | darkdata_keys_super | dbadmin | v_vmart_node0004 |
(10 rows)
```

Default Flex Table View

When you create a flex table, you also create a default view. This default view uses the table name with a `_view` suffix, as listed in the next example, which shows the list of views for `darkdata` and `darkdata1`. If you query the default view, you are prompted to use the [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#) function. This view enables you to update the view after you load data so that it includes all keys and values.

```
=> \dv darkdata*
          List of View Fields
Schema |      View      | Column |      Type      | Size
-----+-----+-----+-----+-----
public | darkdata_view  | status | varchar(124)   | 124
public | darkdata1_view | status | varchar(124)   | 124
(2 rows)
```

For more information, see [Updating Flex Table Views](#).

Flex Functions

There are three sets of functions to support flex tables and extracting data into VMaps. See the following sections for more information:

- Data (helper) functions ([Flex Data Functions Reference](#))
- Extractor functions ([Flex Extractor Functions Reference](#))
- Map functions ([Flex Map Functions Reference](#))

Using Clients with Flex Tables

You can use the Vertica supported client drivers with flex tables as follows:

- To load data into a flex table, you can use the `INSERT` statement or `COPY LOCAL` with the appropriate flex table parser.
- The driver metadata APIs return only real columns. For example, using a `SELECT * FROM myflex;` statement, when `myflex` has a single materialized column (name), returns the `__raw__` and `name` columns. However, it does not return virtual columns from within `__raw__`

_. To access virtual columns and their values, query the associated `flextable_keys` table, just as you would in `vsq`.

Creating Flex Tables

You can create a flex table or an external flex table without column definitions or other parameters. You can use any CREATE TABLE statement parameters you prefer, as usual.

Unsupported CREATE FLEX TABLE Statements

These statements are not currently supported:

- CREATE FLEX TABLE LIKE...

Creating Basic Flex Tables

Here's how to create the table:

```
=> CREATE FLEX TABLE darkdata();  
CREATE TABLE
```

Selecting from the table before loading any data into it reveals its two real columns, `__identity__` and `__raw__`:

```
=> SELECT * FROM darkdata;  
__identity__ | __raw__  
-----+-----  
(0 rows)
```

Below is an example of creating a flex table with a column definition:

```
=> CREATE FLEX TABLE darkdata1(name VARCHAR);  
CREATE TABLE
```

When flex tables exist, you can add new columns (including those with default derived expressions), as described in [Materializing Flex Tables](#).

Creating Temporary Flex Tables

Before you create temporary global and local flex tables, be aware of the following considerations:

- GLOBAL TEMP flex tables are supported. Creating a temporary global flex table results in the `flexable_keys` table and the `flexable_view` having temporary table restrictions for their content.
- LOCAL TEMP flex tables must include at least one column definition. The reason for this requirement is that local temp tables do not support automatically-incrementing data (such as the flex table default `__identity__` column). Creating a temporary local flex table results in the `flexable_keys` table and the `flexable_view` existing in the local temporary object scope.
- LOCAL TEMP views are supported for flex and columnar temporary tables.

For global or local temp flex tables to function correctly, you must also specify the `ON COMMIT PRESERVE ROWS` clause. You must use the `ON COMMIT` clause for the flex table helper functions, which rely on commits. Create a local temp table as follows:

```
=> CREATE FLEX LOCAL TEMP TABLE good(x int) ON COMMIT PRESERVE ROWS;  
CREATE TABLE
```

After creating a local temporary flex table using this approach, you can then load data into the table, create table keys, and a flex table view:

```
=> COPY good FROM '/home/release/KData/bake.json' PARSER fjsonparser();  
Rows Loaded  
-----  
1  
(1 row)  
  
=> select compute_flexable_keys_and_build_view('good');  
compute_flexable_keys_and_build_view  
-----  
Please see v_temp_schema.good_keys for updated keys  
The view good_view is ready for querying  
(1 row)
```

Similarly, you can create global temp tables as follows:

```
=> CREATE FLEX GLOBAL TEMP TABLE good_global(x int) ON COMMIT PRESERVE ROWS;
```

After creating a global temporary flex table using this approach, you can then load data into the table, create table keys, and a flex table view:

```
=> COPY good_global FROM '/home/dbadmin/data/flex/bake_single.json' PARSER fjsonparser();  
Rows Loaded  
-----  
5  
(1 row)  
  
=> SELECT compute_flexable_keys_and_build_view('good_global');
```

```
compute_flexible_keys_and_build_view
```

```
-----  
Please see v_temp_schema.good_keys for updated keys  
The view good_view is ready for querying  
(1 row)
```

Materializing Flex Table Virtual Columns

After you create your flex table and load data, you compute keys from virtual columns. After completing those tasks, you can materialize some keys by promoting virtual columns to real table columns. By promoting virtual columns, you query real columns rather than the raw data.

You can promote one or more virtual columns — materializing those keys from within the `__raw__` data to real columns. Vertica recommends this approach to get the best query performance for all important keys. You don't need to create new columnar tables from your flex table.

Materializing flex table columns results in a hybrid table. Hybrid tables:

- Maintain the convenience of a flex table for loading unstructured data
- Improve query performance for any real columns

If you have only a few columns to materialize, try altering your flex table progressively, adding columns whenever necessary. You can use the `ALTER TABLE . . . ADD COLUMN` statement to do so, just as you would with a columnar table. See [Materializing Flex Tables](#) for ideas about adding columns.

If you want to materialize columns automatically, use the helper function [MATERIALIZED_FLEXIBLE_COLUMNS](#)

Creating Columnar Tables from Flex Tables

You can create a regular Vertica table from a flex table, but you cannot use one flex table to create another.

Typically, you create a columnar table from a flex table after loading data. Then, you specify the virtual column data you want in a regular table, casting virtual columns to regular data types.

(1 row)

1. Check the keys from the `_keys` table for the results of running the helper application:

```
=> SELECT * FROM appLog_keys;
-----+-----+-----
          key_name                               | frequency | data_type_guess
-----+-----+-----
contributors                                   |          8 | varchar(20)
coordinates                                     |          8 | varchar(20)
created_at                                       |          8 | varchar(60)
entities.hashtags                               |          8 | long varbinary(186)
.
.
.
retweeted_status.user.time_zone                 |          1 | varchar(20)
retweeted_status.user.url                       |          1 | varchar(68)
retweeted_status.user.utc_offset                |          1 | varchar(20)
retweeted_status.user.verified                  |          1 | varchar(20)
(125 rows)
```

2. Query from the external flex table view:

```
=> SELECT "user.lang" FROM appLog_view;
 user.lang
-----
it
en
es
en
en
es
tr
en
(12 rows)
```

Note: External tables are fully supported for both flex and columnar tables. However, using external flex (or columnar) tables is less efficient than using flex tables whose data is stored in the Vertica database. Data that is maintained externally requires reloading each time you query.

Creating a Flex Table from Query Results

You can use the `CREATE FLEX TABLE AS` statement to create a flex table from the results of a query.

You can use this statement to create three types of flex tables:

- Flex table with no materialized columns
- Flex table with some materialized columns
- Flex table with all materialized columns

When a flex `__raw__` column is present in the CTAS query, the entire source VMap is carried to the flex table. If the query has matching column names, the key values are overridden.

Note: ORDER BY and segmentation clauses are only passed to the new flex table if the relevant columns are materialized.

Examples

Creating a flex table with no materialized columns from a regular table causes the results of the query to be stored in the `__raw__` column as a VMap.

1. Create a regular table named `pets` with two columns:

```
=> CREATE TABLE pets(age INT, name VARCHAR);  
CREATE TABLE
```

2. Create a flex table named `family_pets` by using the CTAS statement to copy the columns `age` and `name` from the `pets`:

```
=> CREATE FLEX TABLE family_pets() AS SELECT age, name FROM pets;  
CREATE TABLE
```

3. View the new flex table to confirm the operation has been successful and that the columns `age` and `name` have not been materialized.

```
=> \d family_pets;  
List of Fields by Tables  
Schema | Table      | Column      | Type                               | Size  | Default | Not Null |  
Primary Key | Foreign Key  
-----+-----+-----+-----+-----+-----+-----+-----  
public  | family_pets | __identity__ | int                                | 8     |         | t       | f  
|  
public  | family_pets | __raw__      | long varbinary(130000)           | 130000 |         | t       | f  
|  
(2 rows)
```

You can create a flex table with no materialized columns from the results of a query of another flex table. This inserts all the VMaps from the source flex table into the target. This creates a flex table segmented and ordered by the `__identity__` column.

4. Create a flex table named `city_pets` by using the CTAS statement to copy the `age` and `__raw__` columns from `family_pets`:

```
=> CREATE FLEX TABLE city_pets() AS SELECT age, __raw__ FROM family_pets;  
CREATE TABLE
```

5. View the new flex table to confirm that the operation has been successful and the columns `age` and `__raw__` have not been materialized.

```
=> SELECT * FROM city_pets;  
List of Fields by Tables  
Schema | Table | Column | Type | Size | Default | Not Null |  
Primary Key | Foreign Key  
-----+-----+-----+-----+-----+-----+-----+-----  
public | city_pets | __identity__ | int | 8 | | t | f  
|  
public | city_pets | __raw__ | long varbinary(130000) | 130000 | | t | f  
|  
(2 rows)
```

You can create a flex table with some materialized columns. This uses a syntax similar to the syntax for creating columnar tables with some materialized columns. Unlike columnar tables, however, you need to match the number of columns with the columns that are returned by the query. In the following example, our query returns three columns (`amount`, `type`, and `available`), but Vertica only materializes the first two.

6. Create a table named `animals` with three columns, `amount`, `type`, and `available`:

```
=> CREATE TABLE animals(amount INT, type VARCHAR, available DATE);
```

7. Create a flex table named `inventory` with columns `animal_amount` and `animal_type` using the CTAS statement to copy columns `amount`, `type`, and `available` from `animals`.

```
=> CREATE FLEX TABLE inventory(animal_amount, animal_type) AS SELECT amount, type, available  
FROM animals;  
CREATE TABLE
```

8. View the table data to confirm that columns `amount` and `type` have been materialized under the column names `animal_amount` and `animal_type`. Column `available` from `animals` has also been copied over but was not materialized:

```
=> \d inventory
      List of Fields by Tables
Schema | Table | Column          | Type                | Size  | Default | Not Null |
-----+-----+-----+-----+-----+-----+-----+
public | flex3 | __raw__         | long varbinary(130000) | 130000 |         | t        | f
      |      | animal_amount  | int                  | 8      |         | f        | f
      |      | animal_type    | varchar(80)         | 80     |         | f        | f
(3 rows)
```

Notice that including empty parentheses in the statement results in a flex table with no materialized columns:

9. Create a flex table named `animals_for_sale` using the CTAS statement with empty parentheses to copy columns `amount`, `type`, and `available` from `animals` into a pure flex table:

```
=> CREATE FLEX TABLE animals_for_sale() AS SELECT amount, type, available FROM animals;
CREATE TABLE
```

10. View the table data to confirm that no columns were materialized:

```
=>\d animals_for_sale;
      List of Fields by Tables
Schema | Table          | Column          | Type                | Size  | Default | Not Null |
-----+-----+-----+-----+-----+-----+-----+
public | animals_for_sale | __identity__   | int                 | 8     |         | t        |
      |                  | __raw__       | long varbinary(130000) | 130000 |         | t        |
(2 rows)
```

Omitting any parentheses in the statement causes all columns to be materialized:

11. Create a flex table named `animals_sold` using the CTAS statement without parentheses. This copies columns `amount`, `type`, and `available` from `animals` and materialize all columns:

```
=> CREATE FLEX TABLE animals_sold AS SELECT amount, type, available FROM animals;
CREATE TABLE
```

12. View the table data to confirm that all columns were materialized:

```
=> \d animals_sold;
      List of Fields by Tables
```


Bulk Loading Data into Flex Tables

You bulk load data into a flex table with a COPY statement, specifying one of the flex parsers:

- FAVROPARSER
- FCEFPARSER
- FCSVPARSER
- FDELIMITEDPAIRPARSER
- FDELIMITEDPARSER
- FJSONPARSER
- FREGEXPARSER

All flex parsers store the data as a single-value VMap. They reside in the VARBINARY `__raw__` column, which is a real column with a NOT NULL constraint. The VMap is encoded into a single binary value for storage in the `__raw__` column. The encoding places the value strings in a contiguous block, followed by the key strings. Vertica supports null values within the VMap for keys with NULL-specified columns. The key and value strings represent the virtual columns and their values in your flex table.

If a flex table data row is too large to fit in the VARBINARY `__raw__` column, it is rejected. By default, the rejected data and exceptions files are stored in the standard CopyErrorLogs location, a subdirectory of the catalog directory:

```
v_mart_node003_catalog/CopyErrorLogs/trans-STDIN-copy-from-exceptions.1  
v_mart_node003_catalog/CopyErrorLogs/trans-STDIN-copy-rejections.1
```

Flex tables do not copy any rejected data, due to disk space considerations. The rejected data file exists, but it contains only a new line character for every rejected record. The corresponding exceptions file lists the reason why each record was rejected.

You can specify a different path and file for the rejected data and exceptions files. To do so, use the COPY parameters REJECTED DATA and EXCEPTIONS, respectively. You can also save load rejections and exceptions in a table. For more information, see [Bulk-Loading Data](#).

Basic Flex Table Load and Query

Loading data into your flex table is similar to loading data into a regular columnar table. The difference is that you must use the parser argument with one of the flex parsers:

```
=> COPY darkdata FROM '/home/dbadmin/data/tweets_12.json' PARSER fjsonparser();
Rows Loaded
-----
          12
(1 row)
```

Note: You can use many additional COPY parameters as required but not all are supported.

Loading Data into Flex Table Real Columns

If you create a hybrid flex table with one or more real column definitions, COPY evaluates each virtual column key name during data load. For each real column with a name that is identical to a virtual column key name, COPY does the following:

- Loads the keys and values as part of the VMap data in the `__raw__` column
- Automatically populates real columns with the values from their virtual column counterparts

Subsequent data loads continue loading same-name key-value pairs into both the `__raw__` column and the real column.

Note: Over time, storing values in both column types can impact your licensed data limits. For more information about Vertica licenses, see [Managing Licenses](#) in the Administrator's Guide.

For example, continuing with the JSON data:

1. Create a flex table, `darkdata1`, with a column definition of one of the keys in the data you will load:

```
=> CREATE FLEX TABLE darkdata1 ("user.lang" VARCHAR);
CREATE TABLE
```

2. Load data into darkdata1:

```
=> COPY darkdata1 FROM '/test/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
Rows Loaded
-----
                12
(1 row)
```

3. Query the user.lang column of darkdata1. Loading the JSON data file populated the column you defined:

```
=> SELECT "user.lang" FROM darkdata1;
user.lang
-----
es
es
tr
it
en
en
en
en
(12 rows)
```

Empty column rows indicate NULL values. For more information about how NULLs are handled in flex tables, see [NULL Value](#).

4. You can query for other virtual columns (such as "user.name" in darkdata1), with similar results as for "user.lang":

```
=> SELECT "user.name" FROM darkdata1;
user.name
-----
I'm Toasterâ¸
Flu Beach
seydo shi
The End
Uptown gentleman.
~G A B R I E L A â¸
Frederick Danjou
laughing at clouds.
(12 rows)
```

Note: While the results for these two queries are similar, the difference in accessing the keys and their values is significant. Data for "user.lang" has been materialized into a real table column, while "user.name" remains a virtual column. For production-level data usage (rather than test data sets), materializing flex table data improves query performance significantly.

Handling Default Values During Loading

You can create your flex table with a real column, named for a virtual column that exists in your incoming data. For example, if the data you load has a `user.lang` virtual column, define the flex table with that column. You can also specify a default column value when creating the flex table with real columns. The next example shows how to define a real column (`user.lang`), which has a default value from a virtual column (`user.name`):

```
=> CREATE FLEX TABLE darkdata1 ("user.lang" LONG VARCHAR default "user.name");
```

When you load data into your flex table, COPY uses values from the flex table data, ignoring the default column definition. Loading data into a flex table requires [MAPLOOKUP](#) to find keys that match any real column names. A match exists when the incoming data has a virtual column with the same name as a real column. When COPY detects a match, it populates the column with values. COPY returns either a value or NULL for each row, so real columns always have values.

For example, after creating the `darkdata1` flex table, described in the previous example, load data with COPY:

```
=> COPY darkdata1 FROM '/test/vertica/flexible/DATA/tweets_12.json' PARSER fjsonparser();
Rows Loaded
-----
          12
(1 row)
```

If you query the `darkdata1` table after loading, the data shows that the values for the `user.lang` column were extracted:

- From the data being loaded — values for the `user.lang` virtual column
- With NULL — rows without values

In this case, the table column default value for `user.lang` was ignored:

```
=> SELECT "user.lang" FROM darkdata1;
user.lang
-----
it
en
es
en
en
es
tr
```

```
en  
(12 rows)
```

Using COPY to Specify Default Column Values

You can add an expression to a COPY statement to specify default column values when loading data. For flex tables, specifying any column information requires that you list the `__raw__` column explicitly. The following example shows how to use an expression for the default column value. In this case, loading populates the defined `user.lang` column with data from the input data `user.name` values:

```
=> COPY darkdata1(__raw__, "user.lang" as "user.name"::VARCHAR)  
    FROM '/test/vertica/flexibletable/DATA/tweets_12.json' PARSER fjsonparser();  
Rows Loaded  
-----  
          12  
(1 row)  
=> SELECT "user.lang" FROM darkdata1;  
       user.lang  
-----  
laughing at clouds.  
Avita Desai  
I'm Toasterâ€  
Uptown gentleman.  
~G A B R I E L A â€  
Flu Beach  
seydo shi  
The End  
(12 rows)
```

You can specify default values when adding columns, as described in [Altering Flex Tables](#). When you do so, a different behavior results. For more information about using COPY, its expressions and parameters, see [Bulk-Loading Data](#) in the Administrator's Guide and [COPY](#) in the SQL Reference Manual.

Inserting Data into Flex Tables

You can load data into a Vertica flex table using a standard `INSERT` statement, specifying data for one or more columns. When you use `INSERT`, Vertica populates any materialized columns and stores the VMap data in the `__raw__` column.

Vertica provides two ways to use `INSERT` with flex tables:

- `INSERT ... VALUES`
- `INSERT ... SELECT`

Inserting Values into Flex Tables

To insert data values into a flex table, use an `INSERT ... VALUES` statement. If you do not specify any columns in your `INSERT ... VALUES` statement, Vertica positionally assigns values to the real columns of the flex table.

This example shows two ways to insert values into a simple flex table. For both statements, Vertica assigns the values 1 and 'x' to columns a and b, respectively. This example inserts values into the two real columns defined in the flex table:

```
=> CREATE FLEX TABLE flex0 (a INT, b VARCHAR);
CREATE TABLE
=> INSERT INTO flex0 VALUES (1, 'x');
OUTPUT
-----
      1
(1 row)
```

This example inserts values into a flex table without any real columns:

```
=> CREATE FLEX TABLE flex1();
CREATE TABLE
=> INSERT INTO flex1(a,b) VALUES (1, 'x');
OUTPUT
-----
      1
(1 row)
```

For the preceding example, the `__raw__` column contains the inserted data:

```
=> SELECT MapToString(__raw__) FROM flex1;
      MapToString
-----
{
```

```
"a" : "1",  
"b" : "x"  
}  
(1 row)
```

Using INSERT ... SELECT with Flex Tables

Using an INSERT ... SELECT statement with a flex table is similar to using INSERT ... SELECT with a regular table. The SELECT statement returns the data to insert into the target table.

However, Vertica does *not* require that you balance the number of columns and values. If you do not specify a value for a column, Vertica inserts NULL.

In the next example, Vertica copies the a and b values from the flex1 table, and creates columns c, d, e, and f. Because the statement does not specify a value for f, Vertica assigns it a NULL.

```
=> CREATE FLEX TABLE flex2();  
CREATE TABLE  
=> INSERT INTO flex2(a,b) SELECT a,b, '2016-08-10 11:10' c, 'Hello' d, 3.1415 e, f from flex1;  
OUTPUT  
-----  
      1  
(1 row)  
=> SELECT MapToString(__row__) FROM flex2;  
                MapToString  
-----  
{  
"a" : "1",  
"b" : "x",  
"c" : "2016-08-10 11:10",  
  
"d" : "Hello",  
"e" : "3.1415",  
"f" : null  
}  
(1 row)
```

Inserting __row__ Columns into a Flex Table

Inserting a __row__ column into a flex table inserts the entire source VMap into the target table. Vertica does not assign the __row__ column to any target column. Its position in the SELECT statement does not matter.

The following two INSERT statements are equivalent.

```
=> INSERT INTO flex4(a,b) SELECT a, __row__, b FROM flex3;  
=> INSERT INTO flex4(a,b) SELECT a, b, __row__ FROM flex3;
```

Error Handling

Type coercion errors occur only with real columns. The insert operation fails as follows:

```
=> CREATE FLEX TABLE my_table(a INT, b VARCHAR);  
CREATE TABLE  
=> INSERT INTO my_table(a, b) VALUES ('xyz', '5');  
ERROR: Invalid input syntax for integer: "xyz"
```

If you try to insert values into the `__row__` column, the insert fails as follows:

```
=> CREATE FLEX TABLE my_table(a INT, b VARCHAR);  
CREATE TABLE  
=> INSERT INTO my_table(a,b,__row__) VALUES (1,'x','abcdef');  
ERROR 7372: Cannot assign value to "__row__" column
```

See Also

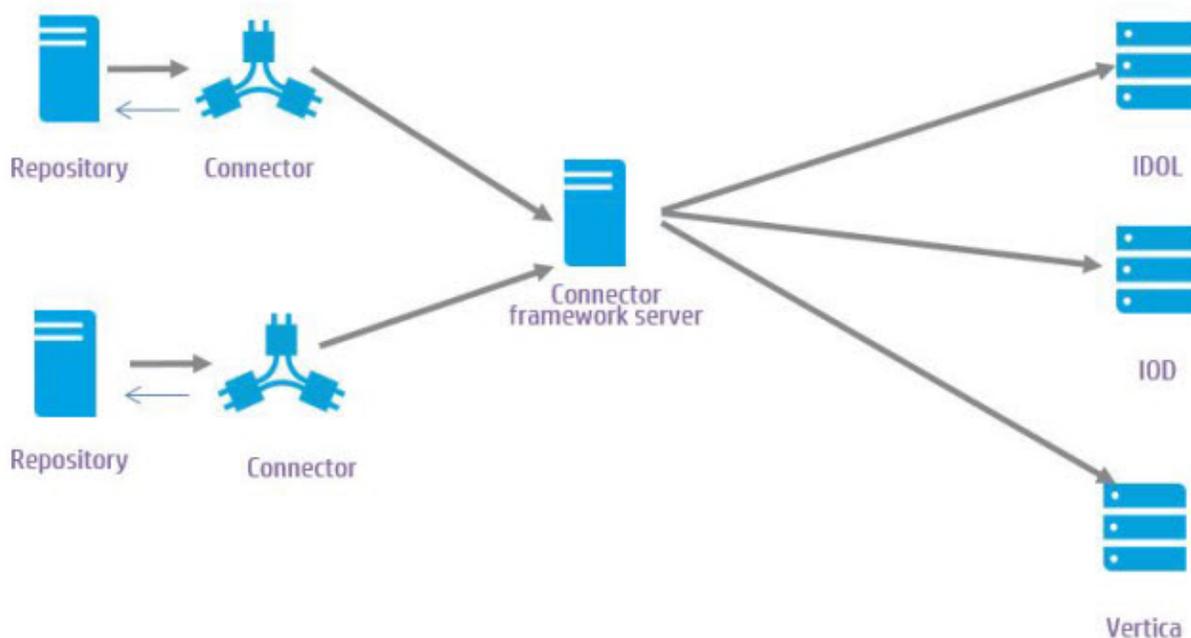
- [INSERT](#)
- [Bulk Loading Data into Flex Tables](#)
- [Data Type Coercion](#)

Using Flex Tables for IDOL Data

You can create flex tables to use with the IDOL Connector Framework Server (CFS) and an ODBC client. The CFS VerticalIndexer module uses the connector to retrieve data. CFS then indexes the data into your Vertica database.

CFS supports many connectors for interfacing to different unstructured file types stored in repositories. Examples of repositories include Microsoft Exchange (email), file systems (including Word documents, images, and videos), Microsoft SharePoint, and Twitter (containing Tweets).

Connectors retrieve and aggregate data from repositories. CFS indexes the data, sending it to IDOL, IDOL OnDemand, or Vertica. The following figure illustrates a basic setup with a repository and a connector.



After you configure CFS and connect it to your Vertica database, the connector monitors the repository for changes and deletions to loaded documents, and for new files not previously added to the server. CFS then updates its server destinations automatically.

To achieve the best query results with ongoing CFS updates and deletes, Micro Focus recommends using live aggregate projections and top-K projections. For more information about how these projections work, and for examples of using them, see [Working with Projections](#) in the Administrator's Guide.

ODBC Connection String for CFS

There are several steps to setting up the CFS VerticalIndexer to load IDOL metadata into your database.

One of the first steps is to add information to the CFS configuration file. To do so, add an Indexing section to the configuration file that specifies the ODBC ConnectionString details.

Successfully loading data requires a valid database user with write permissions to the destination table. Two ODBC connection parameters (UID and PWD) specify the Vertica user and password. The following example shows a sample CFS Indexing section. The section includes a ConnectionString with the basic parameters, including a sample user (UID=fjones) and password (PWD=fjones_password):

```
[Indexing]
IndexerSections=vertica
IndexTimeInterval=30

[vertica]
IndexerType = Library
ConnectionString=Driver=Vertica;Server=123.456.478.900;Database=myDB;UID=fjones;PWD=fjones_password
TableName = marcomm.myFlexTable
LibraryDirectory = ./shared_library_indexers
LibraryName = verticaIndexer
```

For more information about ODBC connection parameters, see [ODBC Configuration Parameters](#).

CFS COPY LOCAL Statement

CFS first indexes and processes metadata from a document repository to add to your database. Then, CFS uses the Indexing information you added to the configuration file to create an ODBC connection. After establishing a connection, CFS generates a standard COPY LOCAL statement, specifying the fjsonparser. CFS loads data directly into your pre-existing flex table with a statement such as the following:

```
=> COPY myFlexTable FROM LOCAL path_to_compressed_temporary_json_file PARSER fjsonparser();
```

```
=> SELECT * FROM myavro;
__identity__ | __raw__
-----+-----
(0 rows)
```

When your IDOL metadata appears in a flex table, you can optionally add new table columns, or materialize other data, as described in [Altering Flex Tables](#).

Using Flex Table Parsers

You can load flex tables with one of several parsers. You can load data using the options that the flex parsers support:

- [Loading Avro Data](#)
- [Loading Common Event Format \(CEF\) Data](#)
- [Loading CSV Data](#)
- [Loading Delimited Data](#)
- [Loading JSON Data](#)
- [Loading Matches from Regular Expressions](#)

Using Flex Parsers for Columnar Tables

You can use any of the flex parsers to load data into columnar tables. Using the flex table parsers to load columnar tables gives you the capability to mix data loads in one table. For example, you can load JSON data in one session and delimited data in another.

Note: For Avro data, you can load only data into a columnar table, not the schema. For flex tables, Avro schema information is required to be embedded in the data.

The following basic examples illustrate how you can use flex parsers with columnar tables.

1. Create a columnar table, `super`, with two columns, `age` and `name`:

```
=> CREATE TABLE super(age INT, name VARCHAR);  
CREATE TABLE
```

2. Enter JSON values from STDIN, using the `fjsonparser()`:

```
=> COPY super FROM stdin PARSER fjsonparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"age": 5, "name": "Tim"}  
>> {"age": 3}  
>> {"name": "Fred"}  
>> {"name": "Bob", "age": 10}
```

```
>> \.
```

3. Query the table to see the values you entered:

```
=> SELECT * FROM super;
 age | name
-----+-----
      | Fred
  10 | Bob
   5 | Tim
   3 |
(4 rows)
```

4. Enter some delimited values from STDIN, using the `fdelimitedparser()`:

```
=> COPY super FROM stdin PARSER fdelimitedparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> name |age
>> Tim|50
>> |30
>> Fred|
>> Bob|100
>> \.
```

5. Query the flex table. Both JSON and delimited data are saved in the same columnar table, `super`.

```
=> SELECT * FROM super;
 age | name
-----+-----
  50 | Tim
  30 |
   3 |
   5 | Tim
 100 | Bob
      | Fred
  10 | Bob
      | Fred
(8 rows)
```

Use the `reject_on_materialized_type_error` parameter to avoid loading data with type mismatch. If `reject_on_materialized_type_error` is set to `false`, the flex parser will accept the data with type mismatch. Consider the following example:

Assume that the CSV file to be loaded has the following sample contents:

```
$ cat json.dat
{"created_by":"system","site_source":"flipkart_india_kol","updated_by":"system1","invoice_id":"INVDPKOL100",
"vendor_id":"VEN15731","total_quantity":12,"created_at":"2012-01-09 23:15:52.0"}
```

```
{"created_by":"system","site_source":"flipkart_india_kol","updated_by":"system2","invoice_id":"INVDPKOL101",  
"vendor_id":"VEN15732","total_quantity":14,"created_at":"hello"}
```

1. Create a columnar table.

```
=> CREATE TABLE hdf5_test (  
  site_source VARCHAR(200),  
  total_quantity int ,  
  vendor_id varchar(200),  
  invoice_id varchar(200),  
  updated_by varchar(200),  
  created_by varchar(200),  
  created_at timestamp  
);
```

2. Load JSON data.

```
=> COPY hdf5_test FROM '/home/dbadmin/json.dat' PARSER fjsonparser() ABORT ON ERROR;  
Rows Loaded  
-----  
2  
(1 row)
```

3. View the contents.

```
=> SELECT * FROM hdf5_test;  
site_source | total_quantity | vendor_id | invoice_id | updated_by | created_by | created_at  
-----+-----+-----+-----+-----+-----+-----  
flipkart_india_kol | 12 | VEN15731 | INVDPKOL100 | system1 | system | 2012-01-09 23:15:52  
flipkart_india_kol | 14 | VEN15732 | INVDPKOL101 | system2 | system |  
(2 rows)
```

4. If `reject_on_materialized_type_error` parameter is set to `true`, you will receive errors when loading the sample JSON data.

```
=> COPY hdf5_test FROM '/home/dbadmin/data/flex/json.dat' PARSER fjsonparser(reject_on_ materialized_type_error=true) ABORT ON ERROR;  
ERROR 2035: COPY: Input record 2 has been rejected (Rejected by user-defined parser)
```

Loading Avro Data

You can load Avro data files into flex tables and columnar tables using the parser, `favroparser`. Before loading, verify that Avro files are encoded in the Avro binary serialization encoding format, described in the [Apache Avro standard](#). The parser also supports Snappy compression. You cannot load Avro data directly from STDIN.

Note: The parser `favroparser` does not support Avro files with separate schema files. The Avro file must have its related schema in the file you are loading.

You can use the following data types and optional parameters for `favroparser`.

The `favroparser` supports two data types:

- [Primitive Data Types for favroparser](#)
- [Complex Data Types for favroparser](#)

Rejecting Data on Materialized Column Type Errors

The `favroparser` has a Boolean parameter, `reject_on_materialized_type_error`. If you set this parameter to `true`, Vertica rejects rows when the input data presents *both* of the following conditions:

- Includes keys matching an existing materialized column
- Has a value that cannot be coerced into the materialized column's data type

Suppose the flex table has a materialized column, `Temperature`, declared as a `FLOAT`. If you try to load a row with a `Temperature` key that has a `VARCHAR` value, `favroparser` rejects the data row.

See Also

- [Using COPY with Data Streaming](#)

Primitive Data Types for favroparser

The `favroparser` supports the following primitive data types:

AVRO Data Type	Vertica Data Type	Value
NULL	NULL Value	No value
boolean	Boolean Data Type	A binary value
int	<code>INTEGER</code>	32-bit

		signed integer
long	INTEGER	64-bit signed integer
float	DOUBLE PRECISION (FLOAT) Synonymous with 64-bit IEEE FLOAT	Single precision (32-bit) IEEE 754 floating-point number
double	DOUBLE PRECISION (FLOAT)	Double precision (64-bit) IEEE 754 floating-point number
bytes	BYTES	Sequence of 8-bit unsigned bytes
string	VARCHAR	Unicode character sequence

Note: Vertica does not have an explicit 4-byte (32-bit integer) or smaller types. Instead, Vertica encoding and compression automatically eliminate the storage overhead of values that require less than 64 bits.

Vertica copies each primitive type into the `__raw__` column of the flex table. In this copy operation, the name of the primitive type becomes a virtual column key with its corresponding value as the value of the virtual column.

If the flex table has materialized columns, `favroparser` loads the primitive data type into the corresponding Vertica type for the column. If the parsing is successful, Vertica copies the data into the materialized column; otherwise, it rejects the row.

Complex Data Types for favroparser

You specify the data type of a record in the Avro file using the `type` parameter for `favroparser`. The `favroparser` supports these complex data types:

- [Records](#)
- [Enums](#)
- [Arrays](#)
- [Maps](#)
- [Unions](#)
- [Fixed](#)

This section describes attributes associated with the complex data types.

Records

Records have the following attributes:

Attribute	Description
name	A JSON string for the name of the record
fields	A JSON array used to list fields. Each field is a JSON object: <ul style="list-style-type: none">• name: A JSON string for the name of the field• type: A JSON object used to define a schema or a JSON string used for naming a record definition

The name of each field is used as a virtual column name. If `flatten_records = true` and several nesting levels are present, Vertica concatenates the record names to create the `key_name`, as follows:

```
{
  "type": "record",
  "name": "Profile",
  "fields" : [
    {"name": "UserName", "type": "string"},
    {"name": "Address", "type": "string"}
  ]
}
```

```
{
  "type": "record",
  "name": "Profile",
  "fields" : [
    {VerticaUser},
    {VerticaUser Address}
  ]
}
```

Vertica creates virtual columns for the records as follows:

Names	Values
UserName	VerticaUser
Address	VerticaUser Address

Enums

Enums (enumerated values) use the type name enum and support the following attributes:

Attribute	Description
name	A JSON string for the name of the enum
symbols	A JSON array used to list symbols as JSON strings. All symbols in an enum must be unique and duplicates are prohibited

Example:

```
{
  "type": "enum",
  "name": "suit",
  "symbols" : ["SPADES", "HEARTS", "DIAMONDS", "CLUBS"]
}
```

Consider the preceding Avro schema with a record that contains a field with the value HEARTS. In this case, the key value pair copied into the `__raw__` column has `suit` as the key and HEARTS as the value.

Arrays

Arrays use the type name `array` and support one attribute:

Attribute	Description
items	The schema of the array's items

For example, declare an array of strings:

```
{"type": "array", "items": "string"}
```

Similar to the capabilities for Records, you can nest and flatten Arrays using `flatten_arrays=true`:

```
{
  "__name__" : "Order",                                <-- artificial __name__ key for record
  "customer_id" : "111222",
  "order_details" : {                                  <-- array of records
    "0" : {                                           <-- array index 0
      "__name__" : "OrderDetail",
      "product_detail" : {
        "__name__" : "Product",
        "price" : "46.21",
        "product_category" : {                        <- array of strings
          "0" : "electronics",
          "1" : "printers",
          "2" : "computers"
        },
        "product_name" : "hp printer X11ew",
        "product_status" : "ONLY_FEW_LEFT"
      }
    },
    "order_id" : "2389646",
    "total" : "132.43"
  }
}
```

Here is the result of flattening the array:

```
{
  "0.order_details.__name__" : "OrderDetail",
  "0.order_details.product_detail.0.product_category" : "electronics",
  "0.order_details.product_detail.1.product_category" : "prnters",
  "0.order_details.product_detail.2.product_category" : "computers",
  "0.order_details.product_detail.__name__" : "Product",
  "0.order_details.product_detail.price" : "46.21",
  "0.order_details.product_detail.product_name" : "hp printer X11ew",
}
```

```
"0.order_details.product_detail.product_status" : "ONLY_FEW_LEFT",  
  "__name__" : "Order",  
  "customer_id" : "111222",  
  "order_id" : "2389646",  
  "total" : "132.43"  
}
```

Maps

Maps use the type name `map` and support one attribute:

Attribute	Description
values	The schema of the map's items

The `favroparser` treats map keys as strings. For example, you can declare the `map` type as a long as follows:

```
{"type": "map", "values": "long"}
```

Similar to Records types, Maps can also be nested and flattened using `flatten_maps=true`.

The `favroparser` inserts key-value pairs from the Avro map as key-value pairs in the `__raw__column`. For an Avro record that has `KeyX` with value `10`, and `KeyY` with value `20`, `favroparser` loads the key-value pairs as virtual columns `KeyX` and `KeyY`, with values `10` and `20`, respectively.

Unions

Vertica uses JSON arrays to represent Avro Unions. Consider this example:

```
{"name": "TransactionID", "type": ["string", "null"]}
```

The field `TransactionID` can be a string or null.

Fixed

Fixed (`fixed`) Avro types support two attributes:

Attribute	Description
name	A string for the name of this

	data type
size	An integer, specifying the number of bytes per value

For example, you can declare a 16-byte quantity:

```
{"type": "fixed", "size": 16, "name": "md5"}
```

With the preceding declaration is the Avro file schema, consider a record that contains a field with the following byte values for the key md5:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
```

The favroparser loads the key value pair as an md5 key with the preceding byte values.

Loading Common Event Format (CEF) Data

Use the flex parser `fcefparser` to load Micro Focus ArcSight or other Common Event Format (CEF) log file data into columnar and flexible tables. For more information, see the [ArcSight Common Event Format \(CEF\) Guide](#).

When you use the parser to load arbitrary CEF-format files, it interprets key names in the data as virtual columns in your flex table. After loading, you can query your CEF data directly, regardless of which set of keys exist in each row. You can also use the associated flex table data and map functions to manage CEF data access.

Create a Flex Table and Load CEF Data

This section uses a sample set of CEF data. All IP addresses have been purposely changed to be inaccurate, and Return characters added for illustration.

To use this sample data, copy the following text and remove all Return characters. Save the file as `CEF_sample.cef`, which is the name used throughout these examples.

```
CEF:0|ArcSight|ArcSight|6.0.3.6664.0|agent:030|Agent [test] type [testalertng] started|Low|  
eventId=1 mrt=1396328238973 categorySignificance=/Normal categoryBehavior=/Execute/Start  
categoryDeviceGroup=/Application catdt=Security Mangement categoryOutcome=/Success  
categoryObject=/Host/Application/Service art=1396328241038 cat=/Agent/Started  
deviceSeverity=Warning rt=1396328238937 fileType=Agent  
cs2=<Resource ID\="3DxKlG0UBABCAA0cXXAZIwA\="\> c6a4=fe80:0:0:0:495d:cc3c:db1a:de71  
cs2Label=Configuration Resource c6a4Label=Agent  
IPv6 Address ahost=SKEELES10 agt=888.99.100.1 agentZoneURI=/All Zones/ArcSight  
System/Private Address Space
```

```
Zones/RFC1918: 888.99.0.0-888.200.255.255 av=6.0.3.6664.0 atz=Australia/Sydney  
aid=3DxK1G0UBABCAA0cXXAZIwA\=\= at=testalertng dvchost=SKEELES10 dvc=888.99.100.1  
deviceZoneURI=/All Zones/ArcSight System/Private Address Space Zones/RFC1918:  
888.99.0.0-888.200.255.255 dtz=Australia/Sydney _cefVer=0.1
```

1. Create a flex table logs:

```
=> CREATE FLEX TABLE logs();  
CREATE TABLE
```

2. Load the sample CEF file, using the flex parser fcefparser:

```
=> COPY logs FROM '/home/dbadmin/data/CEF_sample.cef' PARSER fcefparser();  
Rows Loaded  
-----  
1  
(1 row)
```

3. Use the maptostring() function to see the contents of the logs flex table:

```
=> SELECT maptostring(__raw__) FROM logs;  
maptostring  
-----  
{  
  "_cefver" : "0.1",  
  "agentzoneuri" : "/All Zones/ArcSight System/Private Address  
    Space Zones/RFC1918: 888.99.0.0-888.200.255.255",  
  "agt" : "888.99.100.1",  
  "ahost" : "SKEELES10",  
  "aid" : "3DxK1G0UBABCAA0cXXAZIwA==",  
  "art" : "1396328241038",  
  "at" : "testalertng",  
  "atz" : "Australia/Sydney",  
  "av" : "6.0.3.6664.0",  
  "c6a4" : "fe80:0:0:0:495d:cc3c:db1a:de71",  
  "c6a4label" : "Agent IPv6 Address",  
  "cat" : "/Agent/Started",  
  "catdt" : "Security Mangement",  
  "categorybehavior" : "/Execute/Start",  
  "categorydevicegroup" : "/Application",  
  "categoryobject" : "/Host/Application/Service",  
  "categoryoutcome" : "/Success",  
  "categorysignificance" : "/Normal",  
  "cs2" : "<Resource ID=\"3DxK1G0UBABCAA0cXXAZIwA==\"/>",  
  "cs2label" : "Configuration Resource",  
  "deviceproduct" : "ArcSight",  
  "deviceseverity" : "Warning",  
  "devicevendor" : "ArcSight",  
  "deviceversion" : "6.0.3.6664.0",  
  "devicezoneuri" : "/All Zones/ArcSight System/Private Address Space  
    Zones/RFC1918: 888.99.0.0-888.200.255.255",  
  "dtz" : "Australia/Sydney",
```

```
"dvc" : "888.99.100.1",  
"dvchost" : "SKEELES10",  
"eventid" : "1",  
"filetype" : "Agent",  
"mrt" : "1396328238973",  
"name" : "Agent [test] type [testalertng] started",  
"rt" : "1396328238937",  
"severity" : "Low",  
"signatureid" : "agent:030",  
"version" : "0"  
}  
  
(1 row)
```

Create a Columnar Table and Load CEF Data

This example lets you compare the flex table for CEF data with a columnar table. You do so by creating a new table and load the same `CEF_sample.cef` file used in the preceding flex table example.

1. Create a columnar table, `col_logs`, defining the prefix names that are hard coded in `fcefparser`:

```
=> CREATE TABLE col_logs(version INT,  
devicevendor VARCHAR,  
deviceproduct VARCHAR,  
deviceversion VARCHAR,  
signatureid VARCHAR,  
name VARCHAR,  
severity VARCHAR);  
CREATE TABLE
```

2. Load the sample file into `col_logs`, as you did for the flex table:

```
=> COPY col_logs FROM '/home/dbadmin/data/CEF_sample.cef' PARSER fcefparser();  
Rows Loaded  
-----  
1  
(1 row)
```

3. Query the table. You can find the identical information in the flex table output.

```
=> \x  
Expanded display is on.  
VMart=> SELECT * FROM col_logs;  
-[ RECORD 1 ]+-----  
version      | 0  
devicevendor | ArcSight
```

```
deviceproduct | ArcSight  
deviceversion | 6.0.3.6664.0  
signatureid   | agent:030  
name          | Agent [test] type [testalertng] started  
severity      | Low
```

Compute Keys and Build a Flex Table View

In this example, you use a flex helper function to compute keys and build a view for the logs flex table.

1. Use the `compute_flextable_keys_and_build_view` function to compute keys and populate a view generated from the logs flex table:

```
=> SELECT compute_flextable_keys_and_build_view('logs');  
           compute_flextable_keys_and_build_view  
-----  
Please see public.logs_keys for updated keys  
The view public.logs_view is ready for querying  
(1 row)
```

2. Query the `logs_keys` table to see what the function computed from the sample CEF data:

```
=> SELECT * FROM logs_keys;  
   key_name | frequency | data_type_guess  
-----+-----+-----  
c6a4       |          1 | varchar(60)  
c6a4label  |          1 | varchar(36)  
categoryobject |          1 | varchar(50)  
categoryoutcome |          1 | varchar(20)  
categorysignificance |          1 | varchar(20)  
cs2        |          1 | varchar(84)  
cs2label   |          1 | varchar(44)  
deviceproduct |          1 | varchar(20)  
deviceversion |          1 | varchar(24)  
devicezoneuri |          1 | varchar(180)  
dvchost     |          1 | varchar(20)  
version     |          1 | varchar(20)  
ahost       |          1 | varchar(20)  
art         |          1 | varchar(26)  
at          |          1 | varchar(22)  
cat         |          1 | varchar(28)  
catdt       |          1 | varchar(36)  
devicevendor |          1 | varchar(20)  
dtz         |          1 | varchar(32)  
dvc         |          1 | varchar(24)
```

```
filetype          |          1 | varchar(20)
mrt               |          1 | varchar(26)
_cefver          |          1 | varchar(20)
agentzoneuri     |          1 | varchar(180)
agt              |          1 | varchar(24)
aid              |          1 | varchar(50)
atz              |          1 | varchar(32)
av               |          1 | varchar(24)
categorybehavior |          1 | varchar(28)
categorydevicegroup |          1 | varchar(24)
deviceseverity   |          1 | varchar(20)
eventid          |          1 | varchar(20)
name             |          1 | varchar(78)
rt               |          1 | varchar(26)
severity         |          1 | varchar(20)
signatureid      |          1 | varchar(20)
(36 rows)
```

3. Query several columns from the logs_view:

```
=> \x
Expanded display is on.
VMart=> select version, devicevendor, deviceversion, name, severity, signatureid
        from logs_view;
-[ RECORD 1 ]-+-----
version      | 0
devicevendor | ArcSight
deviceversion | 6.0.3.6664.0
name         | Agent [test] type [testalertng] started
severity     | Low
signatureid  | agent:030
```

Use the fcefparsers Delimiter Parameter

In this example, you use the `fcefparsers delimiter` parameter to query events located in California, New Mexico, and Arizona.

1. Create a new columnar table, CEFDData3:

```
=> CREATE TABLE CEFDData3(eventId INT, location VARCHAR(20));
CREATE TABLE
```

2. Using the `delimiter=' , '` parameter, load some CEF data into the table:

```
=> COPY CEFDData3 FROM stdin PARSER fcefparsers(delimiter=' , ');
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
```

```
>> eventId=1,location=California
>> eventId=2,location=New Mexico
>> eventId=3,location=Arizona
>> \.
```

3. Query the table:

```
=> SELECT eventId, location FROM CEFDData3;
eventId | location
-----+-----
      1 | California
      2 | New Mexico
      3 | Arizona
(3 rows)
```

Loading CSV Data

Use the `fcsvparser` to load data in CSV format (comma-separated values). Since no formal CSV standard exists, Vertica supports the [RFC 4180](#) standard as the default behavior for `fcsvparser`. Other parser parameters simplify various combinations of CSV options into columnar or flex tables. Using `fcsvparser` parses the following CSV data formats:

- **RFC 4180:** The RFC4180 CSV format parser for Vertica flex tables. The parameters for this format are fixed and cannot be changed.
- **Traditional:** The traditional CSV parser lets you specify the parameter values such as delimiter or record terminator. For a detailed list of parameters, please refer the [FCSVPARSER](#)

Using Default Parser Settings

These fixed parameter settings apply to the RCF4180 format.

Note: While you can change the default parser parameter values for Traditional format files, each value must be unique. For example, you can specify an ampersand (&) as a delimiter. If you do, you cannot also use an ampersand for the escape or other parameter value.

Parameter	Data Type	Fixed Value (RCF4180)	Default Value (Traditional)
-----------	-----------	-----------------------	-----------------------------

delimiter	CHAR	,	,
enclosed_by	CHAR	"	"
escape	CHAR	"	\
record_terminator	CHAR	\n or \r\n	\n or \r\n

Use the `type` parameter to indicate either an RFC 4180-compliant file or a traditional-compliant file. You can specify `type` as `RCF4180`. However, you must first verify that the data is compatible with the preceding fixed values for parameters of the RFC4180 format. The default value of the `type` parameter is `RCF4180`.

Loading CSV Data (RFC4180)

Follow these steps to use `fcsvparser` to load data in the RFC4180 CSV data format.

To perform this task, assume that the CSV file to be loaded has the following sample contents:

```
$ more /home/dbadmin/flex/flexData1.csv
sno,name,age,gender
1,John,14,male
2,Mary,23,female
3,Mark,35,male
```

1. Create a flex table:

```
=> CREATE FLEX TABLE csv_basic();
CREATE TABLE
```

2. Load the data from the CSV file using `fcsvparser`:

```
=> COPY csv_basic FROM '/home/dbadmin/flex/flexData1.csv' PARSER fcsvparser();
Rows Loaded
-----
3
(1 row)
```

3. View the data loaded in the flex table:

```
=> SELECT maptostring(__raw__) FROM csv_basic;
maptostring
-----
{
"age" : "14",
"gender" : "male",
"name" : "John",
```

```
"sno" : "1"  
}  
{  
  "age" : "23",  
  "gender" : "female",  
  "name" : "Mary",  
  "sno" : "2"  
}  
{  
  "age" : "35",  
  "gender" : "male",  
  "name" : "Mark",  
  "sno" : "3"  
}  
(3 rows)
```

Loading CSV Data (Traditional)

Follow these steps to use `fcsvparser` to load data in traditional CSV data format using `fcsvparser`.

In this example, the CSV file uses `$` as a delimiter and `#` as a record_terminator. The sample CSV file to load has the following contents:

```
$ more /home/dbadmin/flex/flexData1.csv  
sno$name$age$gender#  
1$John$14$male#  
2$Mary$23$female#  
3$Mark$35$male#
```

1. Create a flex table:

```
=> CREATE FLEX TABLE csv_basic();  
CREATE TABLE
```

2. Load the data in flex table using `fscvparser` with parameters `type='traditional'`, `delimiter='$'` and `record_terminator='#'`:

```
=> COPY csv_basic FROM '/home/dbadmin/flex/flexData2.csv' PARSER fscvparser(type='traditional',  
delimiter='$', record_terminator='#');  
Rows Loaded  
-----  
3  
(1 row)
```

3. View the data loaded in the flex table:

```
=> SELECT maptostring(__raw__) FROM csv_basic;  
maptostring
```

```
-----  
{  
  "age" : "14",  
  "gender" : "male",  
  "name" : "John",  
  "sno" : "1"  
}  
{  
  "age" : "23",  
  "gender" : "female",  
  "name" : "Mary",  
  "sno" : "2"  
}  
{  
  "age" : "35",  
  "gender" : "male",  
  "name" : "Mark",  
  "sno" : "3"  
}  
(3 rows)
```

Rejecting Duplicate Values

You can reject duplicate values using the `reject_on_duplicate=true` option with the `fcsvparser`. The load continues after it rejects a duplicate value. The next example shows how to use this parameter and then displays the specified exception and rejected data files. Saving rejected data to a table, rather than a file, includes both the data and its exception.

```
=> CREATE FLEX TABLE csv_basic();  
CREATE TABLE  
  
=> COPY csv_basic FROM stdin PARSER fcsvparser(reject_on_duplicate=true)  
exceptions '/home/dbadmin/load_errors/except.out' rejected data '/home/dbadmin/load_  
errors/reject.out';  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
  
>> A|A  
>> 1|2  
>> \.  
  
=> \! cat /home/dbadmin/load_errors/reject.out  
A|A  
=> \! cat /home/dbadmin/load_errors/except.out  
COPY: Input record 1 has been rejected (Processed a header row with duplicate keys with  
reject_on_duplicate specified; rejecting.). Please see /home/dbadmin/load_errors/reject.out,  
record 1 for the rejected record.  
COPY: Loaded 0 rows, rejected 1 rows.
```

Rejecting Data on Materialized Column Type Errors

The `fcsvparser` parser has a Boolean parameter, `reject_on_materialized_type_error`. Setting this parameter to `true` causes rows to be rejected if *both* the following conditions exist in the input data:

- Includes keys matching an existing materialized column
- Has a key value that cannot be coerced into the materialized column's data type

The following examples illustrate setting this parameter.

1. Create a table, `reject_true_false`, with two real columns:

```
=> CREATE FLEX TABLE reject_true_false(one int, two int);  
CREATE TABLE
```

2. Load CSV data into the table (from STDIN), using the `fcsvparser` with `reject_on_materialized_type_error=false`. While `false` is the default value, you can specify it explicitly, as shown. Additionally, set the parameter `header=true` to specify the columns for input values:

```
=> COPY reject_true_false FROM stdin PARSER fcsvparser(reject_on_materialized_type_  
error=false,header=true);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> one,two  
>> 1,2  
>> "3","four"  
>> "five",6  
>> 7,8  
>> \.
```

3. Invoke `maptostring` to display the table values after loading data:

```
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;  
maptostring      | one | two  
-----+-----  
{  
"one" : "1",  
"two" : "2"  
}  
|  1 |  2  
{  
"one" : "3",  
"two" : "four"
```

```
}  
| 3 |  
{  
"one" : "five",  
"two" : "6"  
}  
| | 6  
{  
"one" : "7",  
"two" : "8"  
}  
| 7 | 8  
(4 rows)
```

4. Truncate the table to empty the data stored in the table:

```
=> TRUNCATE TABLE reject_true_false;  
TRUNCATE TABLE
```

5. Reload the same data again, but this time, set `reject_on_materialized_type_error=true`:

```
=> COPY reject_true_false FROM stdin PARSER fcsvparser(reject_on_materialized_type_  
error=true,header=true);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> one,two  
>> 1,2  
>> "3","four"  
>> "five",6  
>> 7,8  
>> \.
```

6. Call `maptostring` to display the table contents. Only two rows are currently loaded, whereas the previous results had four rows. The rows having input values with incorrect data type have been rejected:

```
=> SELECT maptostring(__row__), one, two FROM reject_true_false;  
maptostring      | one | two  
-----+-----+-----  
{  
"one" : "1",  
"two" : "2"  
}  
| 1 | 2  
{  
"one" : "7",  
"two" : "8"  
}  
| 7 | 8  
(2 rows)
```

Note: The parser `fcsvparser` uses `null` values if there is a type mismatch and you set the `reject_on_materialized_type_error` parameter to `false`.

Rejecting or Omitting Empty Rows

Valid CSV files can include empty key and value pairs. Such rows are invalid for SQL. You can control the behavior for empty rows by either rejecting or omitting them, using two boolean `FCSVPARSER` parameters:

- `reject_on_empty_key`
- `omit_empty_keys`

The following example illustrates how to set these parameters:

1. Create a flex table:

```
=> CREATE FLEX TABLE csv_basic();  
CREATE TABLE
```

2. Load CSV data into the table (from STDIN), using the `fcsvparser` with `reject_on_empty_key=false`. While `false` is the default value, you can specify it explicitly, as shown. Additionally, set the parameter `header=true` to specify the columns for input values:

```
=> COPY csv_basic FROM stdin PARSER fcsvparser(reject_on_empty_key=false,header=true);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> ,num  
>> 1,2  
>> \.
```

3. Invoke `maptostring` to display the table values after loading data:

```
=>SELECT maptostring(__raw__) FROM csv_basic;  
maptostring  
-----  
{  
  "" : "1",  
  "num" : "2"  
}  
  
(1 row)
```

4. Truncate the table to empty the data stored in the table:

```
=> TRUNCATE TABLE csv_basic;  
TRUNCATE TABLE
```

5. Reload the same data again, but this time, set `reject_on_empty_key=true`:

```
=> COPY csv_basic FROM stdin PARSER fcsvparser(reject_on_empty_key=true,header=true);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> ,num  
>> 1,2  
>> \.
```

6. Call `maptostring` to display the table contents. No rows are loaded because one of the keys is empty:

```
=>SELECT maptostring(__raw__) FROM csv_basic;  
maptostring  
-----  
(0 rows)
```

7. Truncate the table to empty the data stored in the table:

```
=> TRUNCATE TABLE csv_basic;  
TRUNCATE TABLE
```

8. Reload the same data again, but this time, set `omit_empty_keys=true`:

```
=> COPY csv_basic FROM stdin PARSER fcsvparser(omit_empty_keys=true,header=true);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> ,num  
>> 1,2  
>> \.
```

9. Call `maptostring` to display the table contents. One row is now loaded, and the rows with empty keys are omitted:

```
=> SELECT maptostring(__raw__) FROM csv_basic;  
maptostring  
-----  
{  
"num" : "2"  
}  
(1 row)
```

Note: If no header names exist, `fcsvparser` uses a default header of `ucoln`, where `n` is the column offset number. If a table header name and key name match, the parser loads

the column with values associated with the matching key name.

Using the NULL Parameter

Use the COPY NULL metadata parameter with `fcsvparser` to load NULL values into a flex table.

The next example uses this parameter:

1. Create a flex table.

```
=> CREATE FLEX TABLE fcsv(c1 int);  
CREATE TABLE
```

2. Load CSV data in flex table using STDIN and NULL parameter.

```
=> COPY fcsv FROM STDIN PARSER fcsvparser() NULL 'NULL' ;  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> a,b,c1  
>> 10,20,NULL  
>> 20,30,50  
>> 20,30,40  
>> \.
```

3. Use `compute_flexible_keys_and_build_view` function to compute keys and build flex view.

```
=> SELECT compute_flexible_keys_and_build_view('fcsv');  
compute_flexible_keys_and_build_view  
-----  
Please see public.fcsv_keys for updated keys  
The view public.fcsv_view is ready for querying  
(1 row)
```

4. View the flex view and replace the NULL values.

```
=> SELECT * FROM public.fcsv_view;  
a | b | c1  
-----  
20 | 30 | 50  
10 | 20 |  
20 | 30 | 40  
(3 rows)  
  
=> SELECT a,b, ISNULL(c1,-1) from public.fcsv_view;  
a | b | ISNULL  
-----  
20 | 30 | 50  
10 | 20 | -1
```

```
20 | 30 | 40  
(3 rows)
```

Handling Column Headings

The `fcsvparser` lets you specify your own column headings with the `HEADER_NAMES=` parameter. This parameter entirely replaces column names in the CSV source header row.

For example, to use these six column headings for a CSV file you are loading, use the `fcsvparser` parameter as follows:

```
HEADER_NAMES='FIRST, LAST, SOCIAL_SECURITY, TOWN, STATE, COUNTRY'
```

Supplying fewer header names than existing data columns causes `fcsvparser` to use default names after those you supply. Default header names consist of `ucoln`, where n is the column offset number, starting at 0 for the first column. For example, if you supply four header names for a 6-column table, `fcsvparser` supplies the default names `ucol4` and `ucol5`, following the fourth header name you provide.

If you supply more headings than the existing table columns, any additional headings remain unused.

Loading Delimited Data

You can load flex tables with one of two delimited parsers, `fdelimitedparser` or `fdelimitedpairparser`.

- Use `fdelimitedpairparser` when the data specifies column names with the data in each row.

- Use `fdelimitedparser` when the data does not specify column names or has a header row for column names.

This section describes using some options that `fdelimitedpairparser` and `fdelimitedparser` support.

Rejecting Duplicate Values

You can reject duplicate values using the `reject_on_duplicate=true` option with the `fdelimitedparser`. The load continues after it rejects a duplicate value. The next example shows how to use this parameter and then displays the specified exception and rejected data files. Saving rejected data to a table, rather than a file, includes both the data and its exception.

```
=> CREATE FLEX TABLE delim_dupes();
CREATE TABLE

=> COPY delim_dupes FROM stdin PARSER fdelimitedparser(reject_on_duplicate=true)
exceptions '/home/dbadmin/load_errors/except.out' rejected data '/home/dbadmin/load_
errors/reject.out';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> A|A
>> 1|2
>> \.

=> \! cat /home/dbadmin/load_errors/reject.out
A|A
=> \! cat /home/dbadmin/load_errors/except.out
COPY: Input record 1 has been rejected (Processed a header row with duplicate keys with
reject_on_duplicate specified; rejecting.). Please see /home/dbadmin/load_errors/reject.out,
record 1 for the rejected record.
COPY: Loaded 0 rows, rejected 1 rows.
```

Rejecting Materialized Column Type Errors

Both the `fjsonparser` and `fdelimitedparser` parsers have a boolean parameter, `reject_on_materialized_type_error`. Setting this parameter to `true` causes rows to be rejected if *both* the following conditions exist in the input data:

- Includes keys matching an existing materialized column
- Has a value that cannot be coerced into the materialized column's data type

Suppose the flex table has a materialized column, `OwnerPercent`, declared as a `FLOAT`. Trying to load a row with an `OwnerPercent` key that has a `VARCHAR` value causes `fdelimitedparser` to reject the data row.

The following examples illustrate setting this parameter.

1. Create a table, `reject_true_false`, with two real columns:

```
=> CREATE FLEX TABLE reject_true_false(one VARCHAR, two INT);  
CREATE TABLE
```

2. Load JSON data into the table (from STDIN), using the `fjsonparser` with `reject_on_materialized_type_error=false`. While `false` is the default value, the following example specifies it explicitly for illustration:

```
=> COPY reject_true_false FROM stdin PARSER fjsonparser(reject_on_materialized_type_error=false);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"one": 1, "two": 2}  
>> {"one": "one", "two": "two"}  
>> {"one": "one", "two": 2}  
>> \.
```

3. Invoke `maptostring` to display the table values after loading data:

```
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;  
          maptostring      | one | two  
-----+-----+-----  
 {  
  "one" : "one",  
  "two" : "2"  
 }  
 | one | 2  
 {  
  "one" : "1",  
  "two" : "2"  
 }  
 | 1 | 2  
 {  
  "one" : "one",  
  "two" : "two"  
 }  
 | one |  
(3 rows)
```

4. Truncate the table:

```
=> TRUNCATE TABLE reject_true_false;
```

5. Reload the same data again, but this time, set `reject_on_materialized_type_error=true`:

```
=> COPY reject_true_false FROM stdin PARSER fjsonparser(reject_on_materialized_type_error=true);  
Enter data to be copied followed by a newline.
```

```
End with a backslash and a period on a line by itself.  
>> {"one": 1, "two": 2}  
>> {"one": "one", "two": "two"}  
>> {"one": "one", "two": 2}  
>> \.
```

6. Call `maptostring` to display the table contents. Only two rows were loaded, whereas the previous results had three rows:

```
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;  
      maptostring      | one | two  
-----+-----  
 {  
   "one" : "1",  
   "two" : "2"  
 }  
 | 1 | 2  
 {  
   "one" : "one",  
   "two" : "2"  
 }  
 | one | 2  
(2 rows)
```

Loading JSON Data

You can load JSON data into flex or columnar tables. This section describes some examples of using the `fjsonparser` with several of the parser options.

Checking JSON Integrity

Before loading any JSON data, be sure that the data is valid. You can verify JSON data integrity using a web tool such as [JSONLint](#). Copy your JSON data into the tool. If any data is invalid, the tool returns a message similar to the one in this example:

```
Parse error on line 170:...257914002502451200}{  
"id_str": "257  
-----^  
Expecting 'EOF', '}', ',', ']', ''
```

Using `flatten_maps` and `flatten_arrays` Parameters

When loading JSON data, the `fjsonparser` uses the parameters `flatten_maps` and `flatten_arrays` to control how the parser handles the data it is loading. Here are the default settings for these two parameters:

Parameter	Default	Change Default
<code>flatten_maps</code>	TRUE: Flatten all maps.	<code>flatten_maps=FALSE</code>
<code>flatten_arrays</code>	FALSE: Do not flatten arrays.	<code>flatten_arrays=TRUE</code>

You control the default the behavior by using one or both `flatten_` parameters.

For JSON maps, the parser flattens all submaps, separating the levels with a period (`.`). Consider the following input data with a submap:

```
{ grade: { level: 4 } }
```

The default parser behavior results in the following map:

```
{ "grade.level" -> "4" }
```

Note: To use the bracket operators (`[]`) to access deeply nested JSON in VMap data, you must load the data with `flatten_maps=FALSE`, as described in [Querying Nested Data](#).

For JSON arrays, the parser maintains the array. Consider the following input data containing a 2-element array, with values 1 and 2:

```
{ grade: [ 1 2 ] }
```

The default parser behavior results in the following array:

```
{ "grade": { "0" -> "1", "1" -> "2" } }
```

Note: Using the parameters `flatten_maps` and `flatten_arrays` is recursive, and affects all data.

Loading from a Specific Start Point

You can use the `fjsonparser start_point` parameter to load JSON data beginning at a specific key, rather than at the beginning of a file. Data is parsed from after the `start_point` key until the end of the file, or to the end of the first `start_point`'s value. The `fjsonparser` ignores any subsequent instance of the `start_point`, even if that key appears multiple times in the input file. If the input data contains only one copy of the `start_point` key, and that value is a list of JSON elements, the parser loads each element in the list as a row.

This section uses the following sample JSON data, saved to a file (`alphanums.json`):

```
{ "A": { "B": { "C": [ { "d": 1, "e": 2, "f": 3 }, { "g": 4, "h": 5, "i": 6 },  
{ "j": 7, "k": 8, "l": 9 } ] } } }
```

1. Create a flex table, `start_json`:

```
=> CREATE FLEX TABLE start_json();  
CREATE TABLE
```

2. Load `alphanums.json` into `start_json` using the `fjsonparser` without any parameters:

```
=> COPY start_json FROM '/home/dbadmin/data/flex/alphanums.json' PARSER fjsonparser();  
Rows Loaded  
-----  
1  
(1 row)
```

3. Use `maptostring` to see the results of loading all of `alphanums.json`:

```
=> SELECT maptostring(__raw__) FROM start_json;  
maptostring  
-----  
{  
  "A.B.C" : {  
    "0.d" : "1",  
    "0.e" : "2",  
    "0.f" : "3",  
    "1.g" : "4",  
    "1.h" : "5",  
    "1.i" : "6",  
    "2.j" : "7",  
    "2.k" : "8",  
    "2.l" : "9"  
  }  
}
```

```
(1 row)
```

4. Truncate `start_json` and load `alphanums.json` with the `start_point` parameter:

```
=> TRUNCATE TABLE start_json;
TRUNCATE TABLE
=> COPY start_json FROM '/home/dbadmin/data/flex/alphanums.json' PARSE
-> fjsonparser(start_point='B');
Rows Loaded
-----
          1
(1 row)
```

5. Next, call `maptostring` again to compare the results of loading `alphanums.json` from `start_point='B'`:

```
=> SELECT maptostring(__raw__) FROM start_json;
          maptostring
-----
{
  "C" : {
    "0.d" : "1",
    "0.e" : "2",
    "0.f" : "3",
    "1.g" : "4",
    "1.h" : "5",
    "1.i" : "6",
    "2.j" : "7",
    "2.k" : "8",
    "2.l" : "9"
  }
}
(1 row)
```

Parsing From a Start Point Occurrence

If a `start_point` value occurs in multiple locations in your JSON data, you can use the `start_point_occurrence` integer parameter to specify the occurrence at which to start parsing. By defining `start_point_occurrence`, `fjsonparser` begins at the `n`th occurrence of `start_point`.

Controlling Column Name Separators

By default, `fjsonparser` produces column names by concatenating JSON field names with a period (.). You can change the default separator by specifying a different character with the

key_separator parameter.

Handling Special Characters

Some input JSON data can have special characters in field names. You can replace these characters by setting the `suppress_nonalphanumeric_key_chars` to `TRUE`. With this parameter setting, all special characters are converted to an underscore (`_`) character.

Dealing with Invalid JSON Records

If your JSON data is not perfectly formatted, your load may fail due to invalid records. You can use the `RECORD_TERMINATOR` parameter to skip these invalid records if your JSON records are consistently delimited by a character like a line break. Setting a record terminator will allow the `FJSONPARSER` to skip over invalid records and continue parsing the rest of the data.

If your records are not consistently marked by a character, you can use the `COPY` parameter `ERROR TOLERANCE`. `ERROR TOLERANCE` skips entire source files with invalid JSON records, while `RECORD_TERMINATOR` skips individual malformed JSON records. Using both `ERROR TOLERANCE` and `RECORD_TERMINATOR` within the statement will work, but if your records are consistently marked, `RECORD_TERMINATOR` should sufficiently deal with imperfect records.

1. Create a flex table named "fruits".

```
=> CREATE FLEX TABLE fruits();  
CREATE TABLE
```

2. Use the `FJSONPARSER` and call the `RECORD_TERMINATOR` parameter with a termination key of `E'\n'` (which denotes a new line). Insert records, including an invalid record.

```
=> COPY fruits FROM STDIN PARSER FJSONPARSER(RECORD_TERMINATOR=E'\n');  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself  
>> {"name": "orange", "type": "fruit", "color": "orange", "rating": 5 }  
>> {"name": "apple", "type": "fruit", "color": "green" }  
>> {"name": "blueberry", "type": "fruit", "color": "blue", "rating": 10 }  
>> "type": "fruit", "rating": 7 }  
>> {"name": "banana", "type": "fruit", "color": "yellow", "rating": 3 }  
>> \.
```

3. View the flex table using `MAPTOSTRING` to confirm that the invalid record was skipped while the rest of the records were successfully loaded.

```
=> SELECT MAPTOSTRING(__raw__) FROM fruits;  
maptostring  
-----
```

```
{
  "color" : "orange",
  "name" : "orange",
  "rating" : "5",
  "type" : "fruit"
}

{
  "color" : "green",
  "name" : "apple",
  "type" : "fruit"
}

{
  "color" : "blue",
  "name" : "blueberry",
  "rating" : "10",
  "type" : "fruit"
}

{
  "color" : "yellow",
  "name" : "banana",
  "rating" : "3",
  "type" : "fruit"
}
(4 rows)
```

Rejecting Duplicate Values

You can reject duplicate values by using the `reject_on_duplicate=true` option with the `fjsonparser`. The next example uses this option while loading data and then displays the specified exception and rejected data files. Saving rejected data to a table, rather than a file, includes both the data and its exception.

```
=> CREATE FLEX TABLE json_dupes();
CREATE TABLE

=> COPY json_dupes FROM stdin PARSER fjsonparser(reject_on_duplicate=true)
exceptions '/home/dbadmin/load_errors/json_e.out'
rejected data '/home/dbadmin/load_errors/json_r.out';

Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> {"a":"1","a":"2","b":"3"}
>> \.

=> \!cat /home/dbadmin/load_errors/json_e.out
COPY: Input record 1 has been rejected (Rejected by user-defined parser).
Please see /home/dbadmin/load_errors/json_r.out, record 1 for the rejected record.
COPY: Loaded 0 rows, rejected 1 rows.
```

Rejecting Data on Materialized Column Type Errors

Both the `fjsonparser` and `fdelimitedparser` parsers have a Boolean parameter, `reject_on_materialized_type_error`. Setting this parameter to `true` causes rows to be rejected if the input data:

- Includes keys matching an existing materialized column
- Has a key value that cannot be coerced into the materialized column's data type.

The following examples illustrate setting this parameter.

1. Create a table, `reject_true_false`, with two real columns:

```
=> CREATE FLEX TABLE reject_true_false(one VARCHAR, two INT);  
CREATE TABLE
```

2. Load JSON data into the table (from STDIN), using the `fjsonparser` with `reject_on_materialized_type_error=false`. While `false` is the default value, the following example specifies it explicitly for illustration:

```
=> COPY reject_true_false FROM stdin PARSER  
-> fjsonparser(reject_on_materialized_type_error=false);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"one": 1, "two": 2}  
>> {"one": "one", "two": "two"}  
>> {"one": "one", "two": 2}  
>> \.
```

3. Invoke `maptostring` to display the table values after loading data:

```
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;  
maptostring      | one | two  
-----+-----  
{  
  "one" : "one",  
  "two" : "2"  
}  
| one |  2  
{  
  "one" : "1",  
  "two" : "2"  
}  
| 1  |  2  
{  
  "one" : "one",  
  "two" : "two"
```

```
}  
 | one |  
(3 rows)
```

4. Truncate the table:

```
=> TRUNCATE TABLE reject_true_false;
```

5. Reload the same data again, but this time, set `reject_on_materialized_type_error=true`:

```
=> COPY reject_true_false FROM stdin PARSER fjsonparser(reject_on_materialized_type_error=true);  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"one": 1, "two": 2}  
>> {"one": "one", "two": "two"}  
>> {"one": "one", "two": 2}  
>> \.
```

6. Call `maptostring` to display the table contents. Only two rows were loaded, whereas the previous results had three rows:

```
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;  
      maptostring          | one | two  
-----+-----  
 {  
  "one" : "1",  
  "two" : "2"  
 }  
 | 1 | 2  
 {  
  "one" : "one",  
  "two" : "2"  
 }  
 | one | 2  
(2 rows)
```

Rejecting or Omitting Empty Rows

Valid JSON files Micro Focus can include empty key and value pairs, such as this one:

```
{"": 1 "}
```

Such rows are invalid for SQL. To prevent this situation, you can control the behavior for empty rows, either rejecting or omitting them. You do so using two boolean parameters for the parsers `FDELIMITEDPARSER` or `FJSONPARSER`:

- `reject_on_empty_key`
- `omit_empty_keys`

See Also

- [Using COPY with Data Streaming](#)

Loading Matches from Regular Expressions

You can load flex or columnar tables with the matched results of a regular expression, using the `fregexparser`. This section describes some examples of using the options that the flex parsers support.

Sample Regular Expression

These examples use the following regular expression, which searches information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.

This example expression loads any `thread_id` hex value, regardless of whether it has a `0x` prefix, (`<thread_id>(?:0x)?[0-9a-f]+`).

```
'^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)  
(?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)  
-?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]  
\<(?<level>\w+)\> )?(?:<(?<element>\w+)\> @[(?<enode>\w+)]?: )  
?(?<text>.*).'
```

Using Regular Expression Matches for a Flex Table

You can load the results from a regular expression into a flex table, using the `fregexparser`. For a complete example of doing so, see [FREGEXPARSER](#).

Using fregexparser for Columnar Tables

This section illustrates how to load the results of a regular expression used with a sample log file for a Vertica database. By using an external table definition, the section presents an example of using fregexparser to load data into a columnar table. Using a flex table parser for a columnar tables gives you the capability to mix data loads in one table. For example, you can load the results of a regular expression in one session, and JSON data in another.

The following basic examples illustrate this usage.

1. Create a columnar table, vlog, with the following columns:

```
=> CREATE TABLE vlog (  
  "text"          varchar(2322),  
  thread_id      varchar(28),  
  thread_name    varchar(44),  
  "time"         varchar(46),  
  component      varchar(30),  
  level          varchar(20),  
  transaction_id varchar(32),  
  elevel         varchar(20),  
  enode          varchar(34)  
);
```

2. Use COPY to load parts of a log file using the sample regular expression presented above, with the fregexparser. Be sure to remove any line characters from this expression example before trying it yourself:

```
=> COPY v_log FROM '/home/dbadmin/data/flex/vertica.log' PARSER  
FRegexParser(pattern='^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)  
(?<thread_name>[A-Za-z ]+):(?(?<thread_id>(?:0x)?[0-9a-f]+)  
-?(?(?<transaction_id>[0-9a-f])?(?:[(?(?<component>\w+)]  
\<(?(?<level>\w+)\> )?(?:<(?(?<elevel>\w+)\> @[(?(?<enode>\w+)\>)]?: )  
?(?(?<text>.*))') rejected data as table fregex_reject;
```

3. Query the time column:

```
=> SELECT time FROM flogs limit 10;  
      time  
-----  
2014-04-02 04:02:02.613  
2014-04-02 04:02:02.613  
2014-04-02 04:02:02.614  
2014-04-02 04:02:51.008  
2014-04-02 04:02:51.010  
2014-04-02 04:02:51.012  
2014-04-02 04:02:51.012  
2014-04-02 04:02:51.013
```

```
2014-04-02 04:02:51.014  
2014-04-02 04:02:51.017  
(10 rows)
```

Using External Tables with fregexparser

By creating an external columnar table for your Vertica log file, querying the table will return updated log information. The following basic example illustrate this usage.

1. Create a columnar table, `vertica_log`, using the `AS COPY` clause and `fregexparser` to load matched results from the regular expression. For illustrative purposes, this regular expression has new line characters to split long text lines. Remove any line returns before testing with this expression:

```
=> CREATE EXTERNAL TABLE public.vertica_log  
(  
  "text" varchar(2322),  
  thread_id varchar(28),  
  thread_name varchar(44),  
  "time" varchar(46),  
  component varchar(30),  
  level varchar(20),  
  transaction_id varchar(32),  
  elevel varchar(20),  
  enode varchar(34)  
)  
AS COPY  
FROM '/home/dbadmin/data/vertica.log'  
PARSER FRegexParser(pattern='^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)  
(?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)  
-?(?<transaction_id>[0-9a-f]+)?(?:[(?<component>\w+)]  
\<(?!<level>\w+)\> )?(?:<(?!<elevel>\w+)\> @[(?!<enode>\w+)\>]?:  
?(?<text>.*\n)');
```

2. Query from the external table to get updated results:

```
=> SELECT component, thread_id, time FROM vertica_log limit 10;  
component | thread_id | time  
-----+-----+-----  
Init      | 0x16321430 | 2014-04-02 04:02:02.613  
Init      | 0x16321430 | 2014-04-02 04:02:02.614  
Init      | 0x16321430 | 2014-04-02 04:02:02.614
```

(10 rows) | 0x16321430 | 2014-04-02 04:02:02.614

Computing Flex Table Keys

After loading data into a flex table, you can determine the set of keys that exist in the `__raw__` column (the map data). Two helper functions compute keys from flex table map data:

- [COMPUTE_FLEXTABLE_KEYS](#)— Determines which keys exist as virtual columns in the flex map.
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)— Performs the same functionality as [COMPUTE_FLEXTABLE_KEYS](#), additionally building a new view. See also [Updating Flex Table Views](#).

Using COMPUTE_FLEXTABLE_KEYS

During execution, this function calculates the following information for the flex keys table columns:

Column	Description
key_name	The name of the virtual column (key).
frequency	The number of times the key occurs in the map.
data_type_guess	The type guess that the helper functions determine for each distinct key in the map data. The function determines the type of each non-string value, depending on the length of the key, and whether the key includes nested maps. Changing the default value of the <code>EnableBetterFlexTypeGuessing</code> configuration parameter to 0 (OFF) results in the functions determining all flex table keys as string types (<code>[LONG]VARCHAR</code>) or (<code>[LONG] VARBINARY</code>).

Determining Key Data Types

By default, using `COMPUTE_FLEXTABLE_KEYS` determines non-string key values from the `__raw__` column `LONG VARBINARY` type. The non-string keys include these data types (and others listed in [SQL Data Types](#)):

- BOOLEAN
- INTEGER
- FLOAT
- TIMESTAMP
- DATE

Assigning Flex Key Data Types

Use the sample CSV data in this section to compare the results of using or not using the `EnableBetterFlexTypeGuessing` configuration parameter. When the parameter is ON, the function determines key non-string data types in your map data more accurately. The default for the parameter is 1 (ON).

```
Year,Quarter,Region,Species,Grade,Pond Value,Number of Quotes,Available
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,1P,$615.12 ,12,No
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,SM,$610.78 ,12,Yes
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,2S,$596.00 ,20,Yes
2015,1,2 - Northwest Oregon & Willamette,Hemlock,P,$520.00 ,6,Yes
2015,1,2 - Northwest Oregon & Willamette,Hemlock,SM,$510.00 ,6,No
2015,1,2 - Northwest Oregon & Willamette,Hemlock,2S,$490.00 ,14,No
```

To compare the data type assignment results, complete the following steps:

1. Save the CSV data file (here, as `trees.csv`).
2. Create a flex table (`trees`) and load `trees.csv` using the `fcsvparser`:

```
=> CREATE FLEX TABLE trees();
=> COPY trees FROM '/home/dbadmin/tempdat/trees.csv' PARSER fcsvparser();
```

3. Use `COMPUTE_FLEXTABLE_KEYS` with the `trees` flex table.

```
=> SELECT COMPUTE_FLEXTABLE_KEYS('trees');
      COMPUTE_FLEXTABLE_KEYS
-----
Please see public.trees_keys for updated keys
(1 row)
```

4. Query the `trees_keys` table output.:

```
=> SELECT * FROM trees_keys;
  key_name | frequency | data_type_guess
-----+-----+-----
Year      |         6 | Integer
Quarter   |         6 | Integer
Region    |         6 | Varchar(66)
Available |         6 | Boolean
Number of Quotes |         6 | Integer
Grade     |         6 | Varchar(20)
Species   |         6 | Varchar(22)
Pond Value |         6 | Numeric(8,3)
(8 rows)
```

5. Set the `EnableBetterFlexTypeGuessing` parameter to 0 (OFF).
6. Call `COMPUTE_FLEXTABLE_KEYS` with the `trees` flex table again.
7. Query the `trees_keys` table to compare the `data_type_guess` values with the previous results. Without the configuration parameter set, all of the non-string data types are `VARCHARS` of various lengths:

```
=> SELECT * FROM trees_keys;
  key_name | frequency | data_type_guess
-----+-----+-----
Year      |         6 | varchar(20)
Quarter   |         6 | varchar(20)
Region    |         6 | varchar(66)
Available |         6 | varchar(20)
Grade     |         6 | varchar(20)
Number of Quotes |         6 | varchar(20)
Pond Value |         6 | varchar(20)
Species   |         6 | varchar(22)
(8 rows)
```

8. To maintain accurate results for non-string data types, set the `EnableBetterFlexTypeGuessing` parameter back to 1 (ON).

For more information about setting the `EnableBetterFlexTypeGuessing` configuration parameter, see [Setting Flex Table Configuration Parameters](#).

Calculating Key Value Column Widths

The `COMPUTE_FLEXTABLE_KEYS` function determines the column width for keys by determining the length of the largest value for each key, multiplied by the `FlexTableDataTypeGuessMultiplier` factor. For more about this configuration parameter, see [Setting Flex Table Configuration Parameters](#).

The next example shows the results of populating the `_keys` table after creating a flex table (`darkdata1`) and loading data. The column widths are shown in parentheses, where applicable, after the value of the `data_type_guess` column:

```
=> SELECT compute_flextable_keys('darkdata1');
           compute_flextable_keys
-----
Please see public.darkdata1_keys for updated keys
(1 row)

=> SELECT * from darkdata1_keys;
           key_name | frequency | data_type_guess
-----+-----+-----
created_at        |          8 | TimestampTz
delete.status.id_str |          4 | Integer
delete.status.user_id |          4 | Integer
entities.hashtags |          8 | long varbinary(186)
favorited         |          8 | Boolean
id_str            |          8 | Integer
in_reply_to_screen_name |          8 | Varchar(24)
retweeted_status.contributors |          1 | Varchar(20)
retweeted_status.coordinates |          1 | Varchar(20)
retweeted_status.created_at |          1 | TimestampTz
.
.
.
(125 rows)
```

Materializing Flex Tables

Once flex tables exist, you can change the table structure to promote virtual columns to materialized (real) columns. If your table is already a hybrid table, you can change existing real columns and promote other important virtual columns. This section describes some key aspects of promoting columns, adding columns, specifying constraints, and declaring default values. It also presents some differences when loading flex or hybrid tables, compared with columnar tables.

Note: Materializing virtual columns by promoting them to real columns can significantly improve query performance. Vertica recommends that you materialize important virtual columns before running large and complex queries. Promoted columns cause a small decrease in load performance.

Adding Columns to Flex Tables

Add columns to your flex tables to promote virtual columns:

1. Add a real column with the same name as a virtual column (user.name):

```
=> ALTER TABLE darkdata1 ADD COLUMN "user.name" VARCHAR;  
ALTER TABLE
```

2. Load some data into the table.

```
=> COPY darkdata1 FROM '/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();  
Rows Loaded  
-----  
12  
(1 row)
```

3. Query the materialized column. Notice that loading data populates the column automatically. Blank rows indicate no values or NULLs:

```
=> SELECT "user.name" FROM darkdata1;  
user.name  
-----  
I'm Toasterã¥  
Flu Beach  
seydo shi
```

```
The End
Uptown gentleman.
~G A B R I E L A â€¦
Avita Desai
laughing at clouds.
(12 rows)
```

Adding Columns with Default Values

The section [Bulk Loading Data into Flex Tables](#) describes the use of default values, and how Vertica evaluates them during loading. As with all tables, using COPY to load data ignores any column default values.

Note: Adding a table column default expression to a flex table requires casting the column to an explicit data type.

1. Create a `darkdata1` table with a column definition. The following example uses a column name (`talker`) that does not correspond to a virtual column name. Assign a default value with a virtual column name. In this example, the default value for the column `talker` is (`"user.lang"`). Since `user.lang` is a virtual column in the `LONG VARBINARY __raw__` column, you must cast its value to `VARCHAR` to match the `talker` column definition:

```
=> CREATE FLEX TABLE darkdata1(talker VARCHAR default "user.lang":VARCHAR);
CREATE TABLE
```

2. Load some JSON data, specifying the `__raw__` column:

```
=> COPY darkdata1 (__raw__) FROM '/test/vertica/flexible/DATA/tweets_12.json'
PARSER fjsonparser();
Rows Loaded
-----
          12
(1 row)
```

3. Query the `talker` column. Notice that Vertica used the default column value (`"user.lang"`), because you specified `__raw__`. Blank rows indicate no values or NULLs:

```
=> SELECT "talker" FROM darkdata1;
talker
-----
it
```

```
en
es
en
en
es
tr
en
(12 rows)
```

4. Alter the table to add a column with a known virtual column name (`user.name`), assigning the key name as the default value (recommended), and casting it to a VARCHAR:

```
=> ALTER TABLE darkdata1 ADD COLUMN "user.name" VARCHAR default "user.name"::VARCHAR;
ALTER TABLE
```

5. Load data again, this time without `__raw__`:

```
=> COPY darkdata1 FROM '/test/vertica/flexible/DATA/tweets_12.json' PARSER fjsonparser();
```

6. Query the two real columns. Notice that `talker` has no values, because you did not specify the `__raw__` column. The `user.lang` column contains values from the `user.name` virtual column:

```
=> SELECT "talker", "user.name" FROM darkdata1;
talker |      user.name
-----+-----
      | laughing at clouds.
      | Avita Desai
      | I'm Toasterâ
      |
      |
      | Uptown gentleman.
      | ~G A B R I E L A â¸
      | Flu Beach
      |
      | seydo shi
      | The End
(12 rows)
```

7. Load data once more, this time specifying a COPY statement with a default value expression for `user.name`:

```
=> COPY darkdata1 (__raw__, "user.name" as 'QueenElizabeth'::varchar) FROM
'/test/vertica/flexible/DATA/tweets_12.json' PARSER fjsonparser();
Rows Loaded
-----
      12
```

```
(1 row)
```

8. Query once more. Notice that the real column `talker` has its default values (you used `__raw__`). As specified in `COPY`, the `"user.name"` as `'QueenElizabeth'` expression overrode the `user.name` default column value:

```
=> SELECT "talker", "user.name" FROM darkdata1;
talker | user.name
-----+-----
it     | QueenElizabeth
en     | QueenElizabeth
es     | QueenElizabeth
       | QueenElizabeth
       | QueenElizabeth
       | QueenElizabeth
en     | QueenElizabeth
en     | QueenElizabeth
es     | QueenElizabeth
       | QueenElizabeth
tr     | QueenElizabeth
en     | QueenElizabeth
(12 rows)
```

To summarize, you can set a default column value as part of the `ALTER TABLE . . . ADD COLUMN . . .` operation. For materializing columns, the default value should reference the key name of the virtual column (as in `"user.lang"`). Subsequently loading data with a `COPY` value expression overrides the default value of the column definition.

Changing the `__raw__` Column Size

You can change the default size of the `__raw__` column for flex tables you plan to create, the current size of an existing flex table, or both.

To change the default size for the flex table `__raw__` column, use the following database configuration parameter (described in [Setting Flex Table Configuration Parameters](#)):

```
=> ALTER DATABASE mydb SET FlexTableRowSize = 120000;
```

Changing the configuration parameter affects all flex tables you create after making this change.

To change the size of the `__raw__` column in an existing flex table, use the `ALTER TABLE` statement to change the definition of the `__raw__` column:

```
=> ALTER TABLE tester ALTER COLUMN __raw__ SET DATA TYPE LONG VARBINARY(120000);  
ALTER TABLE
```

Note: An error occurs if you try to reduce the `__raw__` column size to a value smaller than any data the column contains.

Changing Flex Table Real Columns

You can make the following changes to the flex table real columns (`__raw__` and `__identity__`), but not to any virtual columns:

Actions	<code>__raw__</code>	<code>__identity__</code>
Change NOT NULL constraints (default)	Yes	Yes
Add primary key and foreign key (PK/FK) constraints	No	Yes
Create projections	No	Yes
Segment	No	Yes
Partition	No	Yes
Specify a user-defined scalar function (UDSF) as a default column expression in <code>ALTER TABLE x ADD COLUMN y</code> statement	No	No

Note: While segmenting and partitioning the `__raw__` column is permitted, it is not recommended due to its long data type. By default, if you not define any real columns, flex tables are segmented on the `__identity__` column.

Dropping Flex Table Columns

There are two considerations about dropping columns:

- You cannot drop the last column in your flex table's sort order.
- If you have not created a flex table with any real columns, or materialized any columns, you cannot drop the `__identity__` column.

Updating Flex Table Views

Creating a flex table also creates a default view to accompany the table. The view has the name of the flex table with an underscore (`_view`) suffix. When you perform a `select` query from the default view, Vertica prompts you to run the helper function, `compute_flextable_keys_and_build_view`:

```
=> \dv dark*

                List of View Fields
Schema |      View      | Column |      Type      | Size
-----+-----+-----+-----+-----
public | darkdata_view  | status | varchar(124)   | 124
public | darkdata1_view | status | varchar(124)   | 124
(2 rows)

=> SELECT * FROM darkdata_view;

                                status
-----
Please run compute_flextable_keys_and_build_view() to update this view to reflect real and
virtual columns in the flex table
(1 row)
```

There are two helper functions that create views:

- [COMPUTE_FLEXTABLE_KEYS](#)— See also [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)— Performs the same functionality as `BUILD_FLEXTABLE_KEYS` but also computes keys. See also [Using COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#).

Using BUILD_FLEXTABLE_VIEW

After computing keys for a flex table ([Computing Flex Table Keys](#)), call this function with one or more arguments. The records under the `key_name` column of the `{flexible}_keys` table are used as view columns, along with any values for the key. If no values exist, the column value is `NULL`.

Regardless of the number of arguments, calling this function replaces the contents of the existing view as follows:

Function Invocation	Results
<code>build_flextable_view ('flexible_table')</code>	Changes the existing view associated with <code>flexible_</code>

Function Invocation	Results
	<i>flexible_table_keystable</i> .
<code>build_flexible_view ('flexible_table', 'view_name')</code>	Changes the view you specify with <i>view_name</i> by using the current contents of the <i>{flexible}_keys</i> table.
<code>build_flexible_view ('flexible_table', 'view_name', 'table_keys')</code>	Changes the view you specify with <i>view_name</i> to the current contents of the <i>flexible_table_keys</i> table. Use this function to change a view of your choice with the contents of the keys of interest.

If you do not specify a *view_name* argument, the default name is the flex table name with a `_view` suffix. For example, if you specify the table `darkdata` as the sole argument to this function, the default view is called `darkdata_view`.

You cannot specify a custom view name with the same name as the default view `flex_table_view`, unless you first drop the default-named view and then create your own view of the same name.

Creating a view stores a definition of the column structure at the time of creation. Thus, if you create a flex table view and then promote virtual columns to real columns, you must rebuild the view. Querying a rebuilt flex table view that has newly promoted real columns produces two results. These results reflect values from both virtual columns in the map data and real columns.

Handling JSON Duplicate Key Names in Views

SQL is a case-insensitive language, so the names `TEST`, `test`, and `TeSt` are identical. JSON data is case sensitive, so that it can validly contain key names of different cases with separate values.

When you build a flex table view, the function generates a warning if it detects same-name keys with different cases in the *{flexible}_keys* table. For example, calling `BUILD_FLEXIBLE_VIEW` or `COMPUTE_FLEXIBLE_KEYS_AND_BUILD_VIEW()` on a flex table with duplicate key names results in these warnings:

```
=> SELECT compute_flexible_keys_and_build_view('dupe');
WARNING 5821: Detected keys sharing the same case-insensitive key name
WARNING 5909: Found and ignored keys with names longer than the maximum column-name length limit

                                compute_flexible_keys_and_build_view
-----
Please see public.dupe_keys for updated keys
The view public.dupe_view is ready for querying
(1 row)
```

While a `{flexible}_keys` table can include duplicate key names with different cases, a view cannot. Creating a flex table view with either of the helper functions consolidates any duplicate key names to one column name, consisting of all lowercase characters. All duplicate key values for that column are saved. For example, if these key names exist in a flex table:

- test
- Test
- tEst

The view will include a virtual column `test` with values from the `test`, `Test`, and `tEst` keys.

Note: The examples in this section include added Return characters to reduce line lengths. The product output may differ.

For example, consider the following query, showing the duplicate `test` key names:

```
=> \x
Expanded display is on.
dbt=> select * from dupe_keys;
-[ RECORD 1 ]-----
key_name      |
TesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttest
TesttestTesttestTesttestTesttest
frequency     | 2
data_type_guess | varchar(20)
-[ RECORD 2 ]-----
key_name      |
TesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttestTesttest
TesttestTesttestTesttestTest12345
frequency     | 2
data_type_guess | varchar(20)
-[ RECORD 3 ]-----
key_name      | test
frequency     | 8
data_type_guess | varchar(20)
-[ RECORD 4 ]-----
```



```
en      | Uptown gentleman.  
en      | The End  
it      | laughing at clouds.  
es      | I'm Toasterâ  
        |  
en      | ~G A B R I E L A â¸  
        |  
en      | Avita Desai  
tr      | seydo shi  
        |  
es      | Flu Beach  
(12 rows)
```

This example shows how to call `build_flextable_view` with the original table and the view you previously created, `dd_view`:

```
=> SELECT build_flextable_view ('darkdata', 'dd_view');  
        build_flextable_view  
-----  
The view public.dd_view is ready for querying  
(1 row)
```

Query the view again. You can see that the function populated the view with the contents of the `darkdata_keys` table. Next, review a snippet from the results, with the `key_name` columns and their values:

```
=> \x  
Expanded display is on.
```

```
=> SELECT * FROM dd_view;  
.   
.   
.   
user.following |   
user.friends_count | 791  
user.geo_enabled | F  
user.id | 164464905  
user.id_str | 164464905  
user.is_translator | F  
user.lang | en  
user.listed_count | 4  
user.location | Uptown..  
user.name | Uptown gentleman.  
.   
.   
.
```

When building views, be aware that creating a view stores a definition of the column structure at the time the view is created. If you promote virtual columns to real columns after building a view, the existing view definition is not changed. Querying this view with a select statement such as the following, returns values from only the `__raw__` column:

```
=> SELECT * FROM myflexable_view;
```

Also understand that rebuilding the view after promoting virtual columns changes the resulting value. Future queries return values from both virtual columns in the map data and from real columns.

Using COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW

Call this function with a flex table to compute Flex table keys (see [Computing Flex Table Keys](#)), and create a view in one step.

Querying Flex Tables

After you create your flex table (with or without additional columns) and load data, you can perform these types of queries:

- SELECT
- COPY
- TRUNCATE
- DELETE

You can use SELECT queries for virtual columns that exist in the `__raw__` column and other real columns in your flex tables. Column names are case insensitive.

Unsupported DDL and DML Statements

You cannot use the following DDL and DML statements with flex tables:

- CREATE TABLE *flex_table* AS...
- CREATE TABLE *flex_table* LIKE...
- SELECT INTO
- UPDATE
- MERGE

Determining Flex Table Data Contents

If you do not know what your flex table contains, two helper functions explore the VMap data to determine its contents. Use these functions to compute the keys in the flex table `__raw__` column and, optionally, build a view based on those keys:

- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)

For more information about these and other helper functions, see [Flex Data Functions Reference](#)

To determine what key value pairs exist as virtual columns:

1. Call the function as follows:

```
=> SELECT compute_flextable_keys('darkdata');
       compute_flextable_keys
-----
Please see public.darkdata_keys for updated keys(1 row)
```

2. View the key names by querying the darkdata_keys table:

```
=> SELECT * FROM darkdata_keys;
```

key_name	frequency	data_type_guess
contributors	8	varchar(20)
coordinates	8	varchar(20)
created_at	8	varchar(60)
entities.hashtags	8	long varbinary(186)
.		
.		
retweeted_status.user.time_zone	1	varchar(20)
retweeted_status.user.url	1	varchar(68)
retweeted_status.user.utc_offset	1	varchar(20)
retweeted_status.user.verified	1	varchar(20)

(125 rows)

Querying Virtual Columns

Continuing with the JSON data example, use a SELECT statement query to explore content from the virtual columns. Then, analyze what is most important to you in case you want to materialize any virtual columns. This example shows how to query some common virtual columns in the VMap data:

```
=> SELECT "user.name", "user.lang", "user.geo_enabled" FROM darkdata1;
       user.name | user.lang | user.geo_enabled
-----+-----+-----
laughing at clouds. | it | T
Avita Desai | en | F
```

```
I'm Toasterâ  | es | T
              |   |   |
              |   |   |
Uptown gentleman. | en | F
~G A B R I E L A â¸ | en | F
Flu Beach        | es | F
              |   |   |
seydo shi       | tr | T
The End         | en | F
(12 rows)
```

Querying Flex Table Keys

If you reference an undefined column ('which_column') in a flex table query, Vertica converts the query to a call to the `maplookup()` function as follows:

```
MAPLOOKUP(__raw__, 'which_column')
```

The `maplookup()` function searches the VMap data for the requested key and returns the following information:

- String values associated with the key for a row.
- NULL if the key is not found.

For more information about handling NULL values, see [MAPCONTAINSKEY\(\)](#).

Using Functions and Casting in Flex Table Queries

You can cast the virtual columns as required and use functions in your `SELECT` statement queries. The next example uses a `SELECT` statement to query the `created_at` and `retweet_count` virtual columns, and to cast their values in the process:

```
=> SELECT "created_at"::TIMESTAMP, "retweet_count"::INT FROM darkdata1 ORDER BY 1 DESC;
  created_at | retweet_count
-----+-----
2012-10-15 14:41:05 |          0
2012-10-15 14:41:05 |          0
2012-10-15 14:41:05 |          0
2012-10-15 14:41:05 |          0
2012-10-15 14:41:05 |          0
```

```
2012-10-15 14:41:05 |           0
2012-10-15 14:41:05 |           0
2012-10-15 14:41:04 |           1
|
|
|
(12 rows)
```

The following query uses the COUNT and AVG functions to determine the average length of text in different languages:

```
=> SELECT "user.lang", count (*), avg(length("text"))::int FROM darkdata1 GROUP BY 1 ORDER BY 2 DESC;
user.lang | count | avg
-----+-----+----
en        |      4 | 42
          |      4 |
es        |      2 | 96
it        |      1 | 50
tr        |      1 | 16
(5 rows)
```

Casting Data Types in a Query

The following query requests the values of the created_at virtual column, without casting to a specific data type:

```
=> SELECT "created_at" FROM darkdata1;
      created_at
-----
Mon Oct 15 18:41:04 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
(12 rows)
```

The next example queries the same virtual column, casting created_at to a TIMESTAMP. Casting results in different output and the regional time:

```
=> SELECT "created_at"::TIMESTAMP FROM darkdata1 ORDER BY 1 DESC;
      created_at
-----
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
```

```
2012-10-15 14:41:05  
2012-10-15 14:41:05  
2012-10-15 14:41:05  
2012-10-15 14:41:04
```

Accessing an Epoch Key

The term *EPOCH* (all uppercase letters) is reserved in Vertica for internal use.

If your JSON data includes a virtual column called epoch, you can query it within your flex table. However, use the `maplookup()` function to do so.

Querying Nested Data

If you load JSON or Avro data with `flatten_arrays=FALSE` (the default), VMap data in the `__raw__` column can contain multiple nested structures. In fact, any VMap JSON or Avro data can contain nested structures. This section describes how best to query such data.

Query VMap Nested Values

To query a nested structure, you can use multiple `maplookup()` functions, one for each level. However, the most efficient method is to use bracket (`[]`) operators.

When parsing or extracting VMap data, the default behavior is to flatten data. Flattened VMap data concatenates key names into one long name, delimiting elements with either the default delimiter (`.`), or a user-defined delimiter character.

To use bracket operators for nested structures in your VMap data, the data must not be flattened. Further, you cannot use bracket operators on any existing, flattened VMap data.

To load or extract VMap data correctly, specify `flatten_maps=FALSE` for `fjsonparser`, `favroparser`, and the `mapjsonextractor()` function.

Note: While bracket operator values look similar to array element specifications, they are strings, not integers. You must enter each nested structure as a string, even if the value is an integer. For example, if the value is 2, specify it as `[' 2 ']`, not `[2]`.

Bracket Operators For Nested JSON

This example uses the following JSON data as an example of nested data. Save this data as `restaurant.json`:

```
{
  "restaurant" : {
    "_name_" : "Bob's pizzeria",
    "cuisine" : "Italian",
    "location" : {"city" : "Cambridge", "zip" : "02140"},
    "menu" : [{"item" : "cheese pizza", "price" : "$8.25"},
              {"item" : "chicken pizza", "price" : "$11.99"},
              {"item" : "spinach pizza", "price" : "$10.50"}]
  }
}
```

Create a flex table, `rests`, and load it with the `restaurant.json` file:

```
=> COPY rests FROM '/home/dbadmin/tempdat/restaurant.json' PARSER fjsonparser (flatten_maps=false);
Rows Loaded
-----
          1
(1 row)
```

After loading your data into a flex table, there are two ways to access nested data using brackets:

- Beginning with the `__raw__` column, followed by the character values in brackets
- Starting with the name of the top-most element, followed by the character values in brackets

Both methods are equally efficient. Here are examples of both:

```
=> SELECT __raw__['restaurant']['location']['city'] FROM rests;
__raw__
-----
Cambridge
(1 row)
=> SELECT restaurant['location']['city'] from rests;
restaurant
-----
Cambridge
(1 row)
```

Bracket Operators for Twitter Data

This example shows how to extract some basic information from Twitter data.

After creating a flex table, `tweets`, and loading in some data, the flex table has a block of tweets.

In the following `SELECT` statement, notice how to specify the `__raw__` column of table `tweets`, followed by the bracket operators to define the virtual columns of interest (`['delete']['status']['user_id']`). This query uses the `COUNT()` function to calculate the number of deleted tweets and outputs 10 results:

```
=> SELECT __raw__['delete']['status']['user_id'] as UserId, COUNT(*) as TweetsDelete from tweets
-> WHERE mapcontainskey(__raw__, 'delete')
-> GROUP BY __raw__['delete']['status']['user_id']
-> ORDER BY TweetsDelete DESC, UserID ASC LIMIT 10;
  UserId | TweetsDelete
-----+-----
106079547 |          4
403474369 |          4
181188657 |          3
```

```
223136123 |      3
770139481 |      3
154602299 |      2
192127653 |      2
215011332 |      2
23321883  |      2
242173898 |      2
(10 rows)
```

Querying Flex Views

Flex tables offer the ability of dynamic schema through the application of query rewriting. Use flex views to support restricted access to flex tables. As with flex tables, each time you use a `select` query on a flex table view, internally, Vertica invokes the `maplookup()` function, to return information on all virtual columns. This query behavior occurs for any flex or columnar table that includes a `__raw__` column.

This example illustrates querying a flex view:

1. Create a flex table.

```
=> CREATE FLEX TABLE twitter();
```

2. Load JSON data into flex table using `fjsonparser`.

```
=> COPY twitter FROM '/home/dbadmin/data/flex/tweets_10000.json' PARSER fjsonparser();  
Rows Loaded  
-----  
10000  
(1 row)
```

3. Create a flex view on top of flex table `twitter` with constraint `retweet_count > 0`.

```
=> CREATE VIEW flex_view AS SELECT __raw__ FROM twitter WHERE retweet_count::int > 0;  
CREATE VIEW
```

4. Query the view. First 5 rows are displayed.

```
=> SELECT retweeted,retweet_count,source FROM (select __raw__ from flex_view) t1 limit 5;  
retweeted | retweet_count | source  
-----+-----+-----  
F          | 1              | <a href="http://blackberry.com/twitter" rel="nofollow">Twitter for  
BlackBerry®</a>  
F          | 1              | web  
F          | 1              | <a href="http://twitter.com/download/iphone" rel="nofollow">Twitter  
for iPhone</a>  
F          | 23             | <a href="http://twitter.com/download/android" rel="nofollow">Twitter  
for Android</a>  
F          | 7              | <a href="http://twitter.com/download/iphone" rel="nofollow">Twitter  
for iPhone</a>  
(5 rows)
```

Listing Flex Tables

You can determine which tables in your database are flex tables by querying the `is_flextable` column of the `v_catalog.tables` system table. For example, use a query such as the following to see all tables with a true (t) value in the `is_flextable` column:

```
=> SELECT table_name, table_schema, is_flextable FROM v_catalog.tables;
  table_name | table_schema | is_flextable
-----+-----+-----
bake1       | public      | t
bake1_keys  | public      | f
del         | public      | t
del_keys    | public      | f
delicious   | public      | t
delicious_keys | public      | f
bake        | public      | t
bake_keys   | public      | f
appLog      | public      | t
appLog_keys | public      | f
darkdata    | public      | t
darkdata_keys | public      | f
(12 rows)
```

Setting Flex Table Configuration Parameters

These configuration parameters affect flex table usage:

- `EnableBetterFlexTypeGuessing`
- `FlexTableRowSize`
- `FlexTableDataTypeGuessMultiplier`

This section presents information about each of the parameters.

To change configuration parameters, and to understand parameter scope, see [Setting Configuration Parameter Values](#) and [General Parameters](#) in the Administrator's Guide.

Using `EnableBetterFlexTypeGuessing`

The `EnableBetterFlexTypeGuessing` configuration parameter is 1 (ON) by default. This setting results in the `COMPUTE_FLEXTABLE_KEYS` or `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW` functions determining data types for string and non-string keys. If you change the configuration parameter to 0 (OFF), the functions determine all flex keys as string types (`[LONG]VARCHAR`) or (`[LONG] VARBINARY`).

You can set this configuration parameter at the database and session level.

For examples of using both settings, see [Computing Flex Table Keys](#).

Specifying the `FlexTableRowSize` Parameter

The `FlexTableRowSize` parameter defines the default width of each flex table `__raw__` column. This column is a `LONG VARBINARY` data type, and contains the map data you load into the table.

Setting this configuration parameter does not affect any existing flex tables, only the default width for new flex tables you create after changing `FlexTableRowSize`.

To change the `__raw__` column width of an existing flex table, use the [ALTER TABLE](#) statement, described in [Materializing Flex Tables](#).

Redefining the FlexTableDataTypeGuessMultiplier

After loading data into a flex table, each key in the `__raw__` column is a `LONG VARBINARY` data type.

When you compute flex table keys with either of the functions, `COMPUTE_FLEXTABLE_KEYS` or `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`, the functions cast each key to the applicable Vertica data type. The key value length is determined by the key's largest value, multiplied by the `FlexTableDataTypeGuessMultiplier` factor. Padding the column width with the multiplier supports future data load key values at least twice the largest key value previously loaded.

The flex keys table that both flex functions populate is used to create the associated flex table view.

Note: The `FlexTableDataTypeGuessMultiplier` value is not used to calculate the width of any real columns in a flex table.

Flex Data Functions Reference

The flex table data helper functions supply information you need to query the data you load. For example, suppose you don't know what keys are available in the map data. If not, you can use the [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#) function to populate a keys table and build a view. The functions aid in querying flex table and other VMap data.

Function	Description
COMPUTE_FLEXTABLE_KEYS	Computes map keys from the map data in a <code>flextable_data</code> table, and populates the <code>flextable_data_keys</code> table with the computed keys. Use this function before building a view.
BUILD_FLEXTABLE_VIEW	Uses the keys in the <code>flextable_data_keys</code> table to create a view definition (<code>flextable_data_view</code>) for the <code>flextable_data</code> table. Use this function after computing flex table keys.
COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW	Performs both of the preceding functions in one call.
MATERIALIZE_FLEXTABLE_COLUMNS	Materializes a default number of columns (50) or more or less, if specified.
RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW	Replaces the <code>flextable_data_keys</code> table and the <code>flextable_data_view</code> , linking both the keys table and the view to the parent flex table.

While the functions are available to all users, they are applicable only to:

- Flex tables
- Associated *flex_table_keys* tables
- Associated *flex_table_view* views

By computing keys and creating views from flex table data, the functions allow you to perform SELECT queries. One function restores the original keys table and view that you specified when you first created the flex table.

Flex Table Dependencies

Each flex table (*flextable*) has two dependent objects:

- *flextable_keys*
- *flextable_view*

While both objects are dependent on their parent table, (*flextable*), you can drop either object independently. Dropping the parent table removes both dependents, without a CASCADE option.

Associating Flex Tables and Views

The helper functions automatically use the dependent table and view if they are internally linked with the parent table. You create both when you create the flex table. You can drop either the `_keys` table or the `_view`, and re-create objects of the same name. However, if you do so, the new objects are not internally linked with the parent flex table.

In this case, you can restore the internal links of these objects to the parent table. To do so, drop the `_keys` table and the `_view` before calling the [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#) function. Calling this function re-creates either, or both, the `_keys` table and the `_view`.

The remaining helper functions perform the tasks described in this section.

BUILD_FLEXTABLE_VIEW

Creates, or re-creates, a view for a default or user-defined `_keys` table, ignoring any empty keys.

Syntax

```
build_flextable_view('flex_table' [ [ , 'view_name' ] [ , 'user_keys_table' ] ])
```

Arguments

<i>flex_table</i>	The flex table name. By default, this function builds or rebuilds a view for the input table with the current contents of the associated <i>flex_table_keys</i> table.
<i>view_name</i>	[Optional] A custom view name. Use this option to build a new view for <i>flex_table</i> with the name you specify.
<i>user_keys_table</i>	[Optional] Specifies a keys table from which to create the view. Use this option if you created a custom <i>user_keys</i> table from the flex table map data, rather than from the default <i>flex_table_keys</i> table. The function builds a view from the keys in <i>user_keys</i> table, rather than from the <i>flex_table_keys</i> table.

Examples

The following examples show how to call `build_flextable_view` with 1, 2, or 3 arguments.

Creating a Default View

To create, or re-create, a default view:

1. Call the function with an input flex table, `darkdata`:

```
=> SELECT build_flextable_view('darkdata');
           build_flextable_view
-----
The view public.darkdata_view is ready for querying
(1 row)
```

The function creates a view with the default name (`darkdata_view`) from the `darkdata_keys` table.

2. Query a key name (`user.id`) from the new or updated view:

```
=> SELECT "user.id" from darkdata_view;
   user.id
-----
 340857907
 727774963
```

```
390498773
288187825
164464905
125434448
601328899
352494946
(12 rows)
```

Creating a Custom Name View

To create, or re-create, a view with a custom name:

1. Call the function with two arguments, an input flex table, `darkdata`, and the name of the view to create, `dd_view`:

```
=> SELECT build_flextable_view('darkdata', 'dd_view');
        build_flextable_view
-----
The view public.dd_view is ready for querying
(1 row)
```

2. Query a key name (`user.lang`) from the new or updated view (`dd_view`):

```
=> SELECT "user.lang" from dd_view;
 user.lang
-----
tr
en
es
en
en
it
es
en
(12 rows)
```

Creating a View from a Custom Keys Table

To create a view from a custom `_keys` table with `build_flextable_view`, the custom table must have the same schema and table definition as the default table (`darkdata_keys`).

Create a custom keys table, using any of these three approaches:

1. Create a columnar table with all keys from the default keys table for a flex table (`darkdata_keys`):

```
=> CREATE table new_darkdata_keys as select * from darkdata_keys;
CREATE TABLE
```

2. Create a columnar table without content (LIMIT 0) from the default keys table for a flex table (darkdata_keys):

```
=> CREATE table new_darkdata_keys as select * from darkdata_keys LIMIT 0;
CREATE TABLE
kdb=> select * from new_darkdata_keys;
  key_name | frequency | data_type_guess
-----+-----+-----
(0 rows)
```

3. Create a columnar table without content (LIMIT 0) from the default keys table, and insert two values ('user.lang', 'user.name') into the key_name column:

```
=> CREATE table dd_keys as select * from darkdata_keys limit 0;
CREATE TABLE
=> insert into dd_keys (key_name) values ('user.lang');
  OUTPUT
  -----
         1
(1 row)
=> INSERT into dd_keys (key_name) values ('user.name');
  OUTPUT
  -----
         1
(1 row)
=> SELECT * from dd_keys;
  key_name | frequency | data_type_guess
-----+-----+-----
 user.lang |           |
 user.name |           |
(2 rows)
```

4. After creating a custom keys table, call build_flexable_view with all arguments (an input flex table, the new view name, the custom keys table):

```
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata', 'dd_view', 'dd_keys');
      build_flexable_view
-----
The view public.dd_view is ready for querying
(1 row)
```

5. Query the new view:

```
=> SELECT * from dd_view;
```

See Also

- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [MATERIALIZER_FLEXTABLE_COLUMNS](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

COMPUTE_FLEXTABLE_KEYS

Computes the virtual columns (keys and values) from the flex table VMap data. Use this function to compute keys without creating an associated table view. To also build a view, use [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#).

The function stores its results in the associated flex_keys table, which has the following columns:

- key_name
- frequency
- data_type_guess

For more information, see [Computing Flex Table Keys](#).

Syntax

```
compute_flextable_keys('flex_table')
```

Arguments

<i>flex_table</i>	The name of a flex table.
-------------------	---------------------------

Using Data Type Guessing

The results of the flex_keys table data_type_guess column depend on the EnableBetterFlexTypeGuessing configuration parameter. By default, the parameter is 1 (ON). This setting results in the function returning all non-string keys in the data_type_guess column as one of the following types (and others listed in [SQL Data Types](#)):

- BOOLEAN
- INTEGER
- FLOAT
- TIMESTAMP
- DATE

Setting the configuration parameter to 0 (OFF), results in the function returning only string types ([LONG]VARCHAR) or ([LONG] VARBINARY) for all values in the data_type_guess column of the flex_keys table .

Assigning Flex Key Data Types

Use the sample CSV data in this section to compare the results of using or not using the EnableBetterFlexTypeGuessing configuration parameter. When the parameter is ON, the function determines key non-string data types in your map data more accurately. The default for the parameter is 1 (ON).

```
Year,Quarter,Region,Species,Grade,Pond Value,Number of Quotes,Available
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,1P,$615.12 ,12,No
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,SM,$610.78 ,12,Yes
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,2S,$596.00 ,20,Yes
2015,1,2 - Northwest Oregon & Willamette,Hemlock,P,$520.00 ,6,Yes
2015,1,2 - Northwest Oregon & Willamette,Hemlock,SM,$510.00 ,6,No
2015,1,2 - Northwest Oregon & Willamette,Hemlock,2S,$490.00 ,14,No
```

To compare the data type assignment results, complete the following steps:

1. Save the CSV data file (here, as trees.csv).
2. Create a flex table (trees) and load trees.csv using the fcsvparser:

```
=> CREATE FLEX TABLE trees();  
=> COPY trees FROM '/home/dbadmin/tempdat/trees.csv' PARSER fcsvparser();
```

3. Use COMPUTE_FLEXTABLE_KEYS with the trees flex table.

```
=> SELECT COMPUTE_FLEXTABLE_KEYS('trees');  
          COMPUTE_FLEXTABLE_KEYS  
-----  
Please see public.trees_keys for updated keys  
(1 row)
```

4. Query the trees_keys table output.:

```
=> SELECT * FROM trees_keys;  
   key_name | frequency | data_type_guess  
-----+-----+-----  
Year        |          6 | Integer  
Quarter     |          6 | Integer  
Region      |          6 | Varchar(66)  
Available   |          6 | Boolean  
Number of Quotes |          6 | Integer  
Grade       |          6 | Varchar(20)  
Species     |          6 | Varchar(22)  
Pond Value  |          6 | Numeric(8,3)  
(8 rows)
```

5. Set the EnableBetterFlexTypeGuessing parameter to 0 (OFF).
6. Call COMPUTE_FLEXTABLE_KEYS with the trees flex table again.
7. Query the trees_keys table to compare the data_type_guess values with the previous results. Without the configuration parameter set, all of the non-string data types are VARCHARS of various lengths:

```
=> SELECT * FROM trees_keys;  
   key_name | frequency | data_type_guess  
-----+-----+-----  
Year        |          6 | varchar(20)  
Quarter     |          6 | varchar(20)  
Region      |          6 | varchar(66)  
Available   |          6 | varchar(20)  
Grade       |          6 | varchar(20)  
Number of Quotes |          6 | varchar(20)  
Pond Value  |          6 | varchar(20)  
Species     |          6 | varchar(22)  
(8 rows)
```

8. To maintain accurate results for non-string data types, set the EnableBetterFlexTypeGuessing parameter back to 1 (ON).

For more information about setting the `EnableBetterFlexTypeGuessing` configuration parameter, see [Setting Flex Table Configuration Parameters](#).

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [MATERIALIZE_FLEXTABLE_COLUMNS](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW

Combines the functionality of [BUILD_FLEXTABLE_VIEW](#) and [COMPUTE_FLEXTABLE_KEYS](#) to compute virtual columns (keys) from the VMap data of a flex table and construct a view. Creating a view with this function ignores empty keys. If you do not need to perform both operations together, use one of the single-operation functions instead.

Syntax

```
compute_flextable_keys_and_build_view('flex_table')
```

Arguments

<i>flex_table</i>	The name of a flex table.
-------------------	---------------------------

Examples

This example shows how to call the function for the darkdata flex table.

```
=> SELECT compute_flexible_keys_and_build_view('darkdata');
       compute_flexible_keys_and_build_view
```

```
-----
Please see public.darkdata_keys for updated keys
The view public.darkdata_view is ready for querying
(1 row)
```

See Also

- [BUILD_FLEXIBLE_VIEW](#)
- [COMPUTE_FLEXIBLE_KEYS](#)
- [MATERIALIZER_FLEXIBLE_COLUMNS](#)
- [RESTORE_FLEXIBLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

MATERIALIZER_FLEXIBLE_COLUMNS

Materializes virtual columns listed as *key_names* in the *flexible_keys* table you compute using either [COMPUTE_FLEXIBLE_KEYS](#) or [COMPUTE_FLEXIBLE_KEYS_AND_BUILD_VIEW](#).

Note: Each column that you materialize with this function counts against the data storage limit of your license. To check your Vertica license compliance, call the `AUDIT()` or `AUDIT_FLEX()` functions.

Syntax

```
materialize_flexible_columns('flex_table' [, n-columns [, keys_table_name] ])
```

Arguments

<i>flex_table</i>	The name of the flex table with columns to materialize. Specifying only the flex table name attempts to materialize up to 50 columns of key names in the default <i>flex_table_keys</i> table. When you use this argument, the function:
-------------------	--

	<ul style="list-style-type: none"> • Skips any columns already materialized • Ignores any empty keys <p>To materialize a specific number of columns, use the optional parameter <code>n_columns</code>, described next.</p>
<i>n-columns</i>	<p>[Optional] The number of columns to materialize. The function attempts to materialize the number of columns from the <code>flex_table_keys</code> table, skipping any columns already materialized.</p> <p>Vertica tables support a total of 1600 columns, which is the largest value you can specify for <code>n-columns</code>. The function orders the materialized results by frequency, descending, <i>key_name</i> when materializing the first <code>n</code> columns.</p>
<i>keys_table_name</i>	<p>[Optional] The name of a <code>flex_keys_table</code> from which to materialize columns. The function:</p> <ul style="list-style-type: none"> • Materializes the number of columns (value of <i>n-columns</i>) from <i>keys_table_name</i> • Skips any columns already materialized • Orders the materialized results by frequency, descending, <i>key_name</i> when materializing the first <code>n</code> columns.

Examples

The following example shows how to call `materialize_flextable_columns` to materialize columns. First, load a sample file of tweets (`tweets_10000.json`) into the flex table `twitter_r`.

After loading data and computing keys for the sample flex table, call `materialize_flextable_columns` to materialize the first four columns:

```
=> COPY twitter_r FROM '/home/release/KData/tweets_10000.json' parser fjsonparser();
Rows Loaded
-----
      10000
(1 row)

=> SELECT compute_flextable_keys ('twitter_r');
           compute_flextable_keys
-----
Please see public.twitter_r_keys for updated keys
(1 row)
```

```
=> select materialize_flextable_columns('twitter_r', 4);
       materialize_flextable_columns
-----
The following columns were added to the table public.twitter_r:
    contributors
    entities.hashtags
    entities.urls
For more details, run the following query:
SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and
table_name = 'twitter_r';

(1 row)
```

The last message in the example recommends querying the `materialize_flextable_columns_results` system table for the results of materializing the columns, as shown:

```
=> SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and
table_name = 'twitter_r';
table_id      | table_schema | table_name |      creation_time      |      key_name      |
status |      message
-----+-----+-----+-----+-----+-----+
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.945484-05 | contributors      |
ADDED | Added successfully
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.94551-05 | entities.hashtags |
ADDED | Added successfully
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.945519-05 | entities.urls     |
ADDED | Added successfully
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.945532-05 | created_at        |
EXISTS | Column of same name already
(4 rows)
```

See the [MATERIALIZE_FLEXTABLE_COLUMNS_RESULTS](#) system table in the SQL Reference Manual.

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

Restores the `_keys` table and the `_view`. The function also links the `_keys` table with its associated flex table, in cases where either table is dropped. The function also indicates whether it restored one or both objects.

Syntax

```
RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('flex_table')
```

Arguments

<i>flex_table</i>	The name of a flex table.
-------------------	---------------------------

Examples

This example shows how to invoke this function with an existing flex table, restoring both the `_keys` table and `_view`:

```
=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
          RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW
-----
The keys table public.darkdata_keys was restored successfully.
The view public.darkdata_view was restored successfully.
(1 row)
```

This example illustrates that the function restored `darkdata_view`, but that `darkdata_keys` did not need restoring:

```
=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
          RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW
-----
The keys table public.darkdata_keys already exists and is linked to darkdata.
The view public.darkdata_view was restored successfully.
(1 row)
```

After restoring the `_keys` table, there is no content. To populate the flex keys, call the [COMPUTE_FLEXTABLE_KEYS](#) meta function.

```
=> SELECT * FROM darkdata_keys;
   key_name | frequency | data_type_guess
-----+-----+-----
(0 rows)
```

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [MATERIALIZATE_FLEXTABLE_COLUMNS](#)

Flex Extractor Functions Reference

The following extractor scalar functions process polystructured data:

- MAPDELIMITEDEXTRACTOR
- MAPJSONEXTRACTOR
- MAPREGEXEXTRACTOR

Each function accepts input data that is:

- Existing database content
- A table
- Returned from an expression
- Entered directly

These functions do not parse data from an external file source. All functions return a single VMap value. The extractor functions can return data with NULL-specified columns.

This section describes each extractor function.

MAPDELIMITEDEXTRACTOR

Extracts data with a delimiter character, and other optional arguments, returning a single VMap value. The USING PARAMETERS phrase specifies optional parameters for the function.

Parameters

delimiter	VARCHAR	Single delimiter character. Default value:
header_names	VARCHAR	[Optional] Specifies header names for columns. Default value: uco1n Where <i>n</i> is the column offset number, starting with 0

		for the first column. The function uses default values if you do not specify values for the <code>header_names</code> parameter.
<code>trim</code>	BOOLEAN	[Optional] Trims white space from header names and field values. Default value: <code>true</code>
<code>treat_empty_val_as_null</code>	BOOLEAN	[Optional] Specifies that empty fields become NULLs, rather than empty strings (' '). Default value: <code>true</code>

Examples

These examples use a short set of delimited data:

```
Name|CITY|New city|State|zip
Tom|BOSTON|boston|MA|01
Eric|Burlington|BURLINGTON|MA|02
Jamie|cambridge|CAMBRIDGE|MA|08
```

To begin, save this data as `delim.dat`.

1. Create a flex table, `dflex`:

```
=> CREATE flex table dflex();
CREATE TABLE
```

2. Use `COPY` to load the `delim.dat` file. Use the flex tables `fdelimitedparser` with the `header='false'` option:

```
=> COPY dflex from '/home/release/kmm/flextables/delim.dat' parser fdelimitedparser
(header='false');
Rows Loaded
-----
                4
(1 row)
```

3. Create a columnar table, `dtab`, with an identity `id` column, a `delim` column, and a column to hold a `VMap`, named `vmap`:

```
=> CREATE TABLE dtab (id IDENTITY(1,1), delim varchar(128), vmap long varbinary(512));
CREATE TABLE
```

4. Use COPY to load the `delim.dat` file into the `dtab` table. For the `mapdelimitedextractor` function, add a header row with `USING PARAMETERS header_names=` option to specify the header row for the sample data, along with `delimiter '!'`:

```
=> COPY dtab(delim, vmap as mapdelimitedextractor(delim
  USING PARAMETERS header_names='Name|CITY|New City|State|Zip')) FROM
'/home/dbadmin/data/delim.dat' DELIMITER '!';

Rows Loaded
-----
                4
(1 row)
```

5. Use `maptostring` for the flex table `dflex` to view the `__raw__` column contents. Notice the default header names in use (`ucol0 – ucol4`), since you specified `header='false'` when you loaded the flex table:

```
=> SELECT maptostring(__raw__) from dflex limit 10;
                maptostring
-----
{
  "ucol0" : "Jamie",
  "ucol1" : "cambridge",
  "ucol2" : "CAMBRIDGE",
  "ucol3" : "MA",
  "ucol4" : "08"
}

{
  "ucol0" : "Name",
  "ucol1" : "CITY",
  "ucol2" : "New city",
  "ucol3" : "State",
  "ucol4" : "zip"
}

{
  "ucol0" : "Tom",
  "ucol1" : "BOSTON",
  "ucol2" : "boston",
  "ucol3" : "MA",
  "ucol4" : "01"
}

{
  "ucol0" : "Eric",
  "ucol1" : "Burlington",
  "ucol2" : "BURLINGTON",
  "ucol3" : "MA",
  "ucol4" : "02"
}

(4 rows)
```

6. Use `maptostring` again, this time with the `dtab` table's `vmap` column. Compare the results of this output to those for the flex table. Note that `maptostring` returns the `header_name` parameter values you specified when you loaded the data:

```
=> SELECT maptostring(vmap) from dtab;
                                     maptostring
-----
{
  "CITY" : "CITY",
  "Name" : "Name",
  "New City" : "New city",
  "State" : "State",
  "Zip" : "zip"
}

{
  "CITY" : "BOSTON",
  "Name" : "Tom",
  "New City" : "boston",
  "State" : "MA",
  "Zip" : "02121"
}

{
  "CITY" : "Burlington",
  "Name" : "Eric",
  "New City" : "BURLINGTON",
  "State" : "MA",
  "Zip" : "02482"
}

{
  "CITY" : "cambridge",
  "Name" : "Jamie",
  "New City" : "CAMBRIDGE",
  "State" : "MA",
  "Zip" : "02811"
}

(4 rows)
```

7. Query the `delim` column to view the contents differently:

```
=> SELECT delim from dtab;
      delim
-----
Name|CITY|New city|State|zip
Tom|BOSTON|boston|MA|02121
Eric|Burlington|BURLINGTON|MA|02482
Jamie|cambridge|CAMBRIDGE|MA|02811
(4 rows)
```

See Also

- [MAPJSONEXTRACTOR](#)
- [MAPREGEXEXTRACTOR](#)

MAPJSONEXTRACTOR

Extracts content of repeated JSON data objects, including nested maps, or data with an outer list of JSON elements. The `USING PARAMETERS` phrase specifies optional parameters for the function. Empty input does not generate a Warning or Error.

Note: The function fails if the output size of the function is greater than 65000.

Parameters

<code>flatten_maps</code>	BOOLEAN	[Optional] Flattens sub-maps within the JSON data, separating map levels with a period (.). Default value: true
<code>flatten_arrays</code>	BOOLEAN	[Optional] Converts lists to sub-maps with integer keys. Lists are not flattened by default. Default value: false
<code>reject_on_duplicate</code>	BOOLEAN	[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues. Default value: false
<code>reject_on_empty_key</code>	BOOLEAN	[Optional] Rejects any row containing a key without a value (<code>reject_on_empty_key=true</code>). Default value: false
<code>omit_empty_keys</code>	BOOLEAN	[Optional] Omits any key from the load data that does not have a value (<code>omit_empty_keys=true</code>).

		Default value: false
start_point	CHAR	[Optional] Specifies the name of a key in the JSON load data at which to begin parsing. The parser ignores all data before the start_point value. The parser processes data after the first instance, and up to the second, ignoring any remaining data. Default value: none

Examples

These examples use the following sample JSON data:

```
{ "id": "5001", "type": "None" }  
{ "id": "5002", "type": "Glazed" }  
{ "id": "5005", "type": "Sugar" }  
{ "id": "5007", "type": "Powdered Sugar" }  
{ "id": "5004", "type": "Maple" }
```

Save this example data as `bake_single.json`, and load that file.

1. Create a flex table, `flexjson`:

```
=> CREATE flex table flexjson();  
CREATE TABLE
```

2. Use `COPY` to load the `bake_single.json` file with the `fjsonparser` parser:

```
=> COPY flexjson from '/home/dbadmin/data/bake_single.json' parser fjsonparser();  
Rows Loaded  
-----  
5  
(1 row)
```

3. Create a columnar table, `coljson`, with an identity column (`id`), a json column, and a column to hold a VMap, called `vmap`:

```
=> CREATE table coljson(id IDENTITY(1,1), json varchar(128), vmap long varbinary(10000));  
CREATE TABLE
```

4. Use `COPY` to load the `bake_single.json` file into the `coljson` table, using the `mapjsonextractor` function:

```
=> COPY coljson (json, vmap AS MapJSONExtractor(json)) FROM '/home/dbadmin/data/bake_
single.json';
Rows Loaded
-----
                5
(1 row)
```

5. Use the `maptostring` function for the flex table `flexjson` to output the `__row__` column contents as strings:

```
=> SELECT maptostring(__row__) from flexjson limit 5;
                maptostring
-----
{
  "id" : "5001",
  "type" : "None"
}

{
  "id" : "5002",
  "type" : "Glazed"
}

{
  "id" : "5005",
  "type" : "Sugar"
}

{
  "id" : "5007",
  "type" : "Powdered Sugar"
}

{
  "id" : "5004",
  "type" : "Maple"
}

(5 rows)
```

6. Use the `maptostring` function again, this time with the `coljson` table's `vmap` column and compare the results. The element order differs:

```
=> SELECT maptostring(vmap) from coljson limit 5;
                maptostring
-----
{
  "id" : "5001",
  "type" : "None"
}

{
  "id" : "5002",
  "type" : "Glazed"
}
```

```
}  
  
{  
  "id" : "5004",  
  "type" : "Maple"  
}  
  
{  
  "id" : "5005",  
  "type" : "Sugar"  
}  
  
{  
  "id" : "5007",  
  "type" : "Powdered Sugar"  
}  
  
(5 rows)
```

See Also

- [MAPDELIMITEXTRACTOR](#)
- [MAPREGEXEXTRACTOR](#)

MAPREGEXEXTRACTOR

Extracts data from a regular expression and returns the results as a VMap. Use the USING PARAMETERS `pattern=` phrase, followed by the regular expression.

Parameters

<code>pattern=</code>	VARCHAR	The regular expression as a string. Default value: An empty string ("").
<code>use_jit</code>	BOOLEAN	[Optional] Uses just-in-time compiling when parsing the regular expression. Default value: false.
<code>record_terminator</code>	VARCHAR	[Optional] The character used to separate input records. Default value: \n.

logline_column	VARCHAR	[Optional] The destination column containing the full string that the regular expression matched. Default value: An empty string ("").
----------------	---------	--

Examples

These examples use the following regular expression, which searches for information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.

This example expression loads any thread_id hex value, regardless of whether it has a 0x prefix, (<thread_id>(?:0x)?[0-9a-f]+).

```
'^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)
(?<thread_name>[A-Za-z ]+):( ?<thread_id>(?:0x)?[0-9a-f]+)
-?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
\<( ?<level>\w+)\> )?(?:<( ?<level>\w+)\> @[(?<enode>\w+)]?: )
?(?<text>.*)'
```

The following examples may include newline characters for display purposes.

1. Create a flex table, flogs:

```
=> CREATE flex table flogs();
CREATE TABLE
```

2. Use COPY to load a sample log file (vertica.log), using the flex table flogparser. Note that this example includes added line characters for displaying long text lines.

```
=> COPY flogs FROM '/home/dbadmin/tempdat/vertica.log' PARSER FREGEXPARSER(pattern='
^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+) (?<thread_name>[A-Za-z ]+):
(?<thread_id>(?:0x)?[0-9a-f])-?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
\<( ?<level>\w+)\> )?(?:<( ?<level>\w+)\> @[(?<enode>\w+)]?: )?(?<text>.*)');
Rows Loaded
-----
81399
(1 row)
```

3. Use MapToString to return the results from calling MapRegexExtractor with a regular expression. The output returns the results of the function in string format.

```
=> SELECT maptostrng(MapregexExtractor(E'2014-04-02 04:02:51.011
TM Moveout:0x2aab9000f860-a000000002067 [Txn] <INFO>
```

```
Begin Txn: a0000000002067 \'Moveout: Tuple Mover\' using PARAMETERS
pattern='^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d.\d+)
(?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
-?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
\<(?!<level>\w+)\> )?(?:<(?!<level>\w+)\> @[?!<enode>\w+])?: )
?(?!<text>.*\')) FROM flogs where __identity__=13;

maptostring
-----
--
{
"component" : "Txn",
"level" : "INFO",
"text" : "Begin Txn: a0000000002067 'Moveout: Tuple Mover'",
"thread_id" : "0x2aab9000f860",
"thread_name" : "TM Moveout",
"time" : "2014-04-02 04:02:51.011",
"transaction_id" : "a0000000002067"
}
(1 row)
```

See Also

- [MAPDELIMITEDEXTRACTOR](#)
- [MAPJSONEXTRACTOR](#)

Flex Map Functions Reference

The flex map functions let you extract and manipulate nested map data.

The first argument of all flex map functions (except `emptymap()` and `mapaggregate()`) takes a VMap. The VMap can originate from the `__raw__` column in a flex table or be returned from a map or extraction function.

All map functions (except for `emptymap()` and `mapaggregate()`), accept either a LONG VARBINARY or a LONG VARCHAR map argument.

In the following example, the outer `maplookup()` function operates on the VMap data returned from the inner `maplookup()` function:

```
=> maplookup(maplookup(ret_map, 'batch'), 'scripts')
```

You can use flex map functions with:

- Flex tables
- Their associated `{flexible}_keys` table
- Automatically generated `{flexible}_view` views.

However, use of these functions does not apply to standard Vertica tables.

EMPTYMAP

Constructs a new VMap with one row but without keys or data. Use this transform function to populate a map without using a flex parser. Instead, you use either from SQL queries or from map data present elsewhere in the database.

Syntax

```
emptymap()
```

Arguments

None

- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPAGGREGATE

Returns a LONG VARBINARY VMap with keys and value pairs supplied from two VARCHAR input columns of an existing columnar table. Using this function requires specifying an over() clause for the source table.

Syntax

```
mapaggregate(source_column1, source_column2)
```

Arguments

<code>source_column1</code>	Table column with values to use as the keys of the key/value pair of the returned VMap data.
<code>source_column2</code>	Table column with values to use as the values in the key/value pair of the returned VMap data.

Examples

This example creates a columnar table `btest`, with two VARCHAR columns, named `keys` and `values`, and adds three sets of values:

```
=> CREATE table btest(keys varchar(10), values varchar(10));  
CREATE TABLE
```

```
=> copy btest from stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> one|1
>> two|2
>> three|3
>> \.
```

After populating the `btest` table, call `mapaggregate()`, using the `over (PARTITION BEST)` clause. This call returns the `raw_map` data:

```
=> SELECT mapaggregate(keys, values) over(PARTITION BEST) from btest;
           raw_map
-----
\001\000\000\000\023\000\000\000\003\000\000\000\020\000\000\000\021\000\000\000\022\000\000\000\000132
\003\000\000\000\020\000\000\000\023\000\000\000\030\000\000\000onethreetwo
(1 row)
```

The next example illustrates using `MAPTOSTRING()` with the returned `raw_map` from `mapaggregate()` to see the values:

```
=> SELECT maptostring(raw_map) from (select mapaggregate(keys, values) over(PARTITION BEST) from
btest) bit;
           maptostring
-----
{
  "one": "1",
  "three": "3",
  "two": "2"
}
(1 row)
```

See Also

- [EMPTYMAP](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)

- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPCONTAINSKEY

Determines whether a VMap contains a virtual column (key). This scalar function returns true (t), if the virtual column exists, or false (f) if it does not. Determining that a key exists before calling `maplookup()` lets you distinguish between NULL returns. The `maplookup()` function uses for both a non-existent key and an existing key with a NULL value.

Syntax

```
mapcontainskey(VMap_data, 'virtual_column_name')
```

Arguments

<code>VMap_data</code>	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
<code>virtual_column_name</code>	The name of the key to check.

Examples

This example shows how to use the `mapcontainskey()` functions with `maplookup()`. View the results returned from both functions. Check whether the empty fields that `maplookup()` returns indicate a NULL value for the row (t) or no value (f):

You can use `mapcontainskey()` to determine that a key exists before calling `maplookup()`. The `maplookup()` function uses both NULL returns and existing keys with NULL values to indicate a non-existent key.

```
=> SELECT maplookup(__row__, 'user.location'), mapcontainskey(__row__, 'user.location') from darkdata
order by 1;
maplookup | mapcontainskey
-----+-----
          | t
          | t
          | t
          | t
Chile     | t
Narnia    | t
Uptown..  | t
chicago  | t
          | f
          | f
          | f
          | f
(12 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)

- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPCONTAINSVALUE

Determines whether a VMap contains a specific value. Use this scalar function to return true (t), if the value exists, or false (f), if it does not.

Syntax

```
mapcontainsvalue(VMap_data, 'virtual_column_value')
```

Arguments

VMap_data	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
virtual_column_value	The value whose existence you want to confirm.

Examples

This example shows how to use `mapcontainsvalue()` to determine whether or not a virtual column contains a particular value. Create a flex table (`ftest`), and populate it with some virtual columns and values. Name both virtual columns one:

```
=> CREATE flex table ftest();  
CREATE TABLE  
=> copy ftest from stdin parser fjsonparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.
```

```
>> {"one":1, "two":2}
>> {"one":"one", "2":"2"}
>> \.
```

Call `mapcontainsvalue()` on the `fctest` map data. The query returns false (f) for the first virtual column, and true (t) for the second, which contains the value one:

```
=> SELECT mapcontainsvalue(__raw__, 'one') from fctest;
mapcontainsvalue
-----
f
t
(2 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPITEMS

Returns information about items in a VMap. Use this transform function with one or more optional arguments to access polystuctured values within the VMap data. This function requires an `OVER()` clause.

Syntax

```
mapItems(VMap_data [, passthrough_arg [,...] ])
```

Arguments

VMap_data	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
passthrough_arg	[Optional] One or more arguments indicating keys within the map data in VMap_data

Examples

The following examples illustrate using `mapItems()` with the `over(PARTITION BEST)` clause.

This example determines the number of virtual columns in the map data using a flex table, labeled `darkmountain`. Query using the `count()` function to return the number of virtual columns in the map data:

```
=> SELECT count(keys) FROM (select mapitems(darkmountain.__raw__) over(PARTITION BEST) from
darkmountain) as a;
  count
-----
      19
(1 row)
```

The next example determines what items exist in the map data:

```
=> SELECT * FROM (select mapitems(darkmountain.__raw__) over(PARTITION BEST) from darkmountain) as a;
  keys      | values
-----+-----
 hike_safety | 50.6
 name       | Mt Washington
 type      | mountain
 height    | 17000
```

```
hike_safety | 12.2
name        | Denali
type        | mountain
height      | 29029
hike_safety | 34.1
name        | Everest
type        | mountain
height      | 14000
hike_safety | 22.8
name        | Kilimanjaro
type        | mountain
height      | 29029
hike_safety | 15.4
name        | Mt St Helens
type        | volcano
(19 rows)
```

Directly Query a Key Value in a VMap

Review the following JSON input file, `simple.json`. In particular, notice the array called `three_Array`, and its four values:

```
{
  "one": "one",
  "two": 2,
  "three_Array":
  [
    "three_One",
    "three_Two",
    3,
    "three_Four"
  ],
  "four": 4,
  "five_Map":
  {
    "five_One": 51,
    "five_Two": "Fifty-two",
    "five_Three": "fifty three",
    "five_Four": 54,
    "five_Five": "5 x 5"
  },
  "six": 6
}
```

1. Create a flex table, `mapper`:

```
=> CREATE flex table mapper();
CREATE TABLE
```

1. Load `simple.json` into the flex table `mapper`:

```
=> COPY mapper from '/home/dbadmin/data/simple.json' parser fjsonparser (flatten_arrays=false,
flatten_maps=false);
Rows Loaded
-----
```

```
1  
(1 row)
```

2. Call `mapkeys` on the flex table's `__raw__` column to see the flex table's keys, but not the key submaps. The return values indicate `three_Array` as one of the virtual columns:

```
=> SELECT mapkeys(__raw__) over() from mapper;  
keys  
-----  
five_Map  
four  
one  
six  
three_Array  
two  
(6 rows)
```

3. Call `mapitems` on flex table `mapper` with `three_Array` as a pass-through argument to the function. The call returns these array values:

```
=> SELECT __identity__, mapitems(three_Array) OVER(PARTITION BY __identity__) from mapper;  
__identity__ | keys | values  
-----+-----+-----  
1 | 0 | three_One  
1 | 1 | three_Two  
1 | 2 | 3  
1 | 3 | three_Four  
(4 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)

- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPKEYS

Returns the virtual columns (and values) present in any VMap data. This transform function requires an `over(PARTITION BEST)` clause.

Syntax

```
mapkeys(VMap_data)
```

Arguments

VMap_data	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
-----------	--

Examples

Determine Number of Virtual Columns in Map Data

This example shows how to create a query, using an `over(PARTITION BEST)` clause with a flex table, `darkdata` to find the number of virtual column in the map data. The table is populated with JSON tweet data.

```
=> SELECT count(keys) FROM (SELECT mapkeys(darkdata.__raw__) over(PARTITION BEST) from darkdata) as a;  
count  
-----
```

```
550  
(1 row)
```

Query Ordered List of All Virtual Columns in the Map

This example shows a snippet of the return data when you query an ordered list of all virtual columns in the map data:

```
=> SELECT * FROM (SELECT mapkeys(darkdata.__raw__) over(PARTITION BEST) from darkdata) as a;  
  keys  
-----  
contributors  
coordinates  
created_at  
delete.status.id  
delete.status.id_str  
delete.status.user_id  
delete.status.user_id_str  
entities.hashtags  
entities.media  
entities.urls  
entities.user_mentions  
favorited  
geo  
id  
.  
.  
.  
user.statuses_count  
user.time_zone  
user.url  
user.utc_offset  
user.verified  
(125 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)

- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPKEYSINFO

Returns virtual column information from a given map. This transform function requires an `over(PARTITION BEST)` clause.

Syntax

```
mapkeysinfo(VMap_data)
```

Arguments

<code>VMap_data</code>	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
------------------------	--

Returns

This function is a superset of the [MAPKEYS\(\)](#) function. It returns the following information about each virtual column:

Column	Description
<code>keys</code>	The virtual column names in the raw data.

Column	Description
length	The data length of the key name, which can differ from the actual string length.
type_oid	The OID type into which the value should be converted. Currently, the type is always 116 for a LONG VARCHAR, or 199 for a nested map that is stored as a LONG VARBINARY.
row_num	The number of rows in which the key was found.
field_num	The field number in which the key exists.

Examples

This example shows a snippet of the return data you receive if you query an ordered list of all virtual columns in the map data:

```
=> SELECT * FROM (SELECT mapkeysinfo(darkdata.__raw__) over(PARTITION BEST) from darkdata) as a;
      keys | length | type_oid | row_num | field_num
-----+-----+-----+-----+-----
contributors | 0 | 116 | 1 | 0
coordinates | 0 | 116 | 1 | 1
created_at | 30 | 116 | 1 | 2
entities.hashtags | 93 | 199 | 1 | 3
entities.media | 772 | 199 | 1 | 4
entities.urls | 16 | 199 | 1 | 5
entities.user_mentions | 16 | 199 | 1 | 6
favorited | 1 | 116 | 1 | 7
geo | 0 | 116 | 1 | 8
id | 18 | 116 | 1 | 9
id_str | 18 | 116 | 1 | 10
.
.
.
delete.status.id | 18 | 116 | 11 | 0
delete.status.id_str | 18 | 116 | 11 | 1
delete.status.user_id | 9 | 116 | 11 | 2
delete.status.user_id_str | 9 | 116 | 11 | 3
delete.status.id | 18 | 116 | 12 | 0
delete.status.id_str | 18 | 116 | 12 | 1
delete.status.user_id | 9 | 116 | 12 | 2
delete.status.user_id_str | 9 | 116 | 12 | 3
(550 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPLOOKUP

Returns single-key values from VMAP data. This scalar function returns a LONG VARCHAR, with values, or NULL if the virtual column does not have a value.

Using `maplookup` is case insensitive to virtual column names. To avoid loading same-name values, set the `fjsonparser parser reject_on_duplicate` parameter to `true` when data loading.

You can control the behavior for non-scalar values in a VMAP (like arrays), when loading data with the `fjsonparser` or `favroparser` parsers and its `flatten-arrays` argument. See [Loading JSON Data](#) and the [FJSONPARSER](#) reference.

For information about using `maplookup()` to access nested JSON data, see [Querying Nested Data](#).

Syntax

```
maplookup (VMap_data, 'virtual_column_name' [USING PARAMETERS [case_
sensitive={false | true}] [, buffer_size=n] ] )
```

Parameters

<code>VMap_data</code>	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
<code>virtual_column_name</code>	<p>The name of the virtual column whose values this function returns.</p>
<code>buffer_size</code>	<p>[Optional parameter] Specifies the maximum length (in bytes) of each value returned for <code>virtual_column_name</code>. To return all values for <code>virtual_column_name</code>, specify a <code>buffer_size</code> equal to or greater than (<code>=></code>) the number of bytes for any returned value. Any returned values greater in length than <code>buffer_size</code> are rejected.</p> <p>Default value: 0 (No limit on <code>buffer_size</code>)</p>
<code>case_sensitive</code>	<p>[Optional parameter]</p> <p>Specifies whether to return values for <code>virtual_column_name</code> if keys with different cases exist.</p> <p>Example:</p> <pre>(... USING PARAMETERS case_sensitive=true)</pre> <p>Default value: false</p>

Examples

This example returns the values of one virtual column, `user.location`:

```
=> SELECT maplookup(__raw__, 'user.location') FROM darkdata order by 1;
maplookup
-----
Chile
Nesnia
Uptown
.
.
chicago
(12 rows)
```

Using maplookup buffer_size

Use the `buffer_size=` parameter to indicate the maximum length of any value that `maplookup` returns for the virtual column you specify. If none of the returned key values can be greater than `n` bytes, use this parameter to allocate `n` bytes as the `buffer_size`.

For the next example, save this JSON data to a file, `simple_name.json`:

```
{
  "name": "sierra",
  "age": "63",
  "eyes": "brown",
  "weapon": "doggie"
}
{
  "name": "janis",
  "age": "10",
  "eyes": "blue",
  "weapon": "humor"
}
{
  "name": "ben",
  "age": "43",
  "eyes": "blue",
  "weapon": "sword"
}
{
  "name": "jen",
  "age": "38",
  "eyes": "green",
  "weapon": "shopping"
}
```

1. Create a flex table, `logs`.
2. Load the `simple_name.json` data into `logs`, using the `fjsonparser`. Specify the `flatten_arrays` option as `True`:

```
=> COPY logs FROM '/home/dbadmin/data/simple_name.json'
PARSER fjsonparser(flatten_arrays=True);
```

3. Use `maplookup` with `buffer_size=0` for the `logs` table name key. This query returns all of the values:

```
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=0) FROM logs;
MapLookup
-----
sierra
ben
janis
jen
(4 rows)
```

4. Next, call `maplookup()` three times, specifying the `buffer_size` parameter as 3, 5, and 6, respectively. Now, `maplookup()` returns values with a byte length less than or equal to (\leq) `buffer_size`:

```
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=3) FROM logs;
MapLookup
-----

ben

jen
(4 rows)
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=5) FROM logs;
MapLookup
-----

janis
jen
ben
(4 rows)
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=6) FROM logs;
MapLookup
-----
sierra
janis
jen
ben
(4 rows)
```

Disambiguate Empty Output Rows

This example shows how to interpret empty rows. Using `maplookup` without first checking whether a key exists can be ambiguous. When you review the following output, 12 empty rows, you cannot determine whether a `user.location` key has:

- A non-NULL value
- A NULL value
- No value

```
=> SELECT maplookup(__raw__, 'user.location') FROM darkdata;
maplookup
-----

(12 rows)
```

To disambiguate empty output rows, use the `mapcontainskey()` function in conjunction with `maplookup()`. When `maplookup` returns an empty field, the corresponding value from `mapcontainskey` indicates `t` for a NULL or other value, or `f` for no value.

The following example output using both functions lists rows with NULL or a name value as `t`, and rows with no value as `f`:

```
=> SELECT maplookup(__raw__, 'user.location'), mapcontainskey(__raw__, 'user.location')
from darkdata order by 1;
maplookup | mapcontainskey
-----+-----
          | t
          | t
          | t
          | t
Chile     | t
Nesnia   | t
Uptown   | t
chicago | t
          | f >>>>>>>>No value
          | f >>>>>>>>No value
          | f >>>>>>>>No value
          | f >>>>>>>>No value

(12 rows)
```

Check for Case-Sensitive Virtual Columns

You can use `maplookup()` with the `case_sensitive` parameter to return results when key names with different cases exist.

1. Save the following sample content as a JSON file. This example saves the file as `repeated_key_name.json`:

```
{
  "test": "lower1"
}
```


- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPSIZE

Returns the number of virtual columns present in any VMap data. Use this scalar function to determine the size of keys.

Syntax

```
mapsize(VMap_data)
```

Arguments

<code>VMap_data</code>	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
------------------------	--

Examples

This example shows the returned sizes from the number of keys in the flex table `darkmountain`:

```
=> SELECT mapsize(__raw__) FROM darkmountain;
mapsize
-----
      3
      4
      4
      4
      4
(5 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPTOSTRING

Recursively builds a string representation VMap data, including nested JSON maps. Use this transform function to display the VMap contents in a readable LONG VARCHAR format. Use `maptostring` to see how map data is nested before querying virtual columns with `mapvalues()`.

Syntax

```
maptostring(VMap_data [using parameters canonical_json={true  
| false}])
```

Arguments

VMap_data	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
-----------	---

Parameters

canonical_json	<p><i>=bool</i> [Optional parameter]</p> <p>Produces canonical JSON output by default, using the first instance of any duplicate keys in the map data.</p> <p>Use this parameter as other UDF parameters, preceded by <code>using</code> parameters, as shown in the examples. Setting this argument to <code>false</code> maintains the previous behavior of <code>maptostring()</code> and returns same-name keys and their values.</p> <p>Default value: <code>canonical-json=true</code></p>
----------------	---

Examples

The following example shows how to create a sample flex table, `darkdata` and load JSON data from STDIN. By calling `maptostring()` twice with both values for the `canonical_json` parameter, you can see the different results on the flex table `__raw__` column data.

1. Create sample table:

```
=> CREATE flex table darkdata();  
CREATE TABLE
```

2. Load sample JSON data from STDIN:

```
=> COPY darkdata FROM stdin parser fjsonparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"aaa": 1, "aaa": 2, "AAA": 3, "bbb": "aaa\"bbb"}  
>> \.
```

3. Call `maptostring()` with its default behavior using canonical JSON output, and then review the flex table contents. The function returns the first duplicate key and its value (`"aaa" : "1"`) but omits remaining duplicate keys (`"aaa" : "2"`):

```
=> SELECT maptostring (__raw__) FROM darkdata;  
          maptostring  
-----  
{  
  "AAA" : "3",  
  "aaa" : "1",  
  "bbb" : "aaa\"bbb"  
}  
(1 row)
```

4. Next, call `maptostring()` with using parameters `canonical_json=false`). This time, the function returns the first duplicate keys and their values:

```
=> SELECT maptostring(__raw__ using parameters canonical_json=false) FROM darkdata;  
          maptostring  
-----  
{  
  "aaa": "1",  
  "aaa": "2",  
  "AAA": "3",  
  "bbb": "aaa\"bbb"
```

```
}  
(1 row)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPVALUES

Returns a string representation of the top-level values from a VMap. This transform function requires an `over()` clause.

Syntax

```
mapvalues(VMap_data)
```

Arguments

VMap_data	<p>The VMap from which values should be returned. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
-----------	--

Examples

The following example shows how to query a `darkmountain` flex table, using an `over()` clause (in this case, the `over(PARTITION BEST)` clause) with `mapvalues()`.

```
=> SELECT * FROM (select mapvalues(darkmountain.__raw__) over(PARTITION BEST) from darkmountain) as
a;
  values
-----
29029
34.1
Everest
mountain
29029
15.4
Mt St Helens
volcano
17000
12.2
Denali
mountain
14000
22.8
Kilimanjaro
mountain
50.6
Mt Washington
mountain
(19 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVERSION](#)

MAPVERSION

Returns the version or invalidity of any map data. This scalar function returns the map version (such as 1) or -1, if the map data is invalid.

Syntax

```
mapversion(VMap_data)
```

Arguments

VMap_data	The VMap data either from a <code>__raw__</code> column in a flex table or from the data returned from a map function such as <code>maplookup()</code> .
-----------	--

Examples

The following example shows how to use `mapversion()` with the `darkmountainflex` table, returning `mapversion 1` for the flex table `map data`:

```
=> SELECT mapversion(__raw__) FROM darkmountain;  
mapversion  
-----  
          1  
          1  
          1  
          1  
          1  
(5 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)

Flex Parsers Reference

Vertica supports several parsers to load different types of data into flex tables:

- [FAVROPARSER](#)
- [FCEFPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FDELIMITEDPARSER](#)
- [FJSONPARSER](#)
- [FREGEXPARSER](#)

Unlike with columnar tables, you must specify which parser to use when loading flex tables. You can use each flex parser to load the parser's associated type of data into columnar tables.

All parsers store the data as a single Vmap in the LONG VARBINARY `__raw__` column. If a data row is too large to fit in the column, it is rejected. Vertica supports null values for loading data with NULL-specified columns.

For information about how you can use each type of flex parser, see [Using Flex Table Parsers](#)

FAVROPARSER

Parses data from an Avro file. The input file must use binary serialization encoding. Use this parser to load data into columnar, flex, and hybrid tables.

Note: The parser `favroparser` does not support Avro files with separate schema files. The Avro file must have its related schema in the file you are loading.

Parameters

<code>flatten_arrays</code>	BOOLEAN	[Optional] Flattens all Avro arrays. Key names are concatenated with nested levels.
-----------------------------	---------	---

		Default value: false (Arrays are not flattened.)
flatten_maps	BOOLEAN	[Optional] Flattens all Avro maps. Key names are concatenated with nested levels. Default value: true
flatten_records	BOOLEAN	[Optional] Flattens all Avro records. Key names are concatenated with nested levels. Default value: true
reject_on_materialized_type_error	BOOLEAN	[Optional] Indicates whether to reject any row value for a materialized column that the parser cannot coerce into a compatible data type. See Using Flex Table Parsers . Default value: false

Examples

This example shows how to create and load a flex table with Avro data using favroparser. After loading the data, you can query virtual columns.

1. Create a flex table for Avro data, avro_basic:

```
=> CREATE FLEX TABLE avro_basic();  
CREATE TABLE
```

2. Use the favroparser to load the data from an Avro file (weather.avro).

```
=> COPY avro_basic FROM '/home/dbadmin/data/flexcsv/weather.avro' PARSER favroparser();  
Rows Loaded  
-----  
5  
(1 row)
```

3. Query virtual columns from the avro flex table:

```
=> SELECT station, temp, time FROM avro_basic;  
station | temp | time  
-----+-----+-----  
mohali | 0 | -619524000000  
lucknow | 22 | -619506000000
```

```
norwich | -11 | -619484400000  
ams     | 111 | -655531200000  
baddi   | 78  | -655509600000  
(5 rows)
```

For more information, see [Loading Avro Data](#).

See Also

- [FCEFPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FJSONPARSER](#)
- [FREGEXPARSER](#)

FCEFPARSER

Parses Micro Focus ArcSight Common Event Format (CEF) log files. The `fcefparser` loads values directly into any table column with a column name that matches a source data key. The parser stores the data loaded into a flex table in a single VMap.

Parameters

<code>delimiter</code>	CHAR	[Optional] Specifies a single-character delimiter. Default value: ' '
<code>record_terminator</code>	CHAR	[Optional] Specifies a single-character record terminator. Default value: <code>newline</code>
<code>trim</code>	BOOLEAN	[Optional] Trims white space from

		header names and key values. Default value: true
reject_on_unescaped_delimiter	BOOLEAN	[Optional] Determines whether to reject rows containing unescaped delimiters. The CEF standard does not permit them. Default value: false

Examples

The following example illustrates creating a sample flex table for CEF data, with two real columns, eventId and priority.

1. Create a flex table cefdata:

```
=> create flex table cefdata();  
CREATE TABLE
```

2. Load some basic CEF data, using the flex parser fcefparser:

```
=> copy cefdata from stdin parser fcefparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> CEF:0|ArcSight|ArcSight|2.4.1|machine:20|New alert|High|  
>> \.
```

3. Use the maptostring() function to view the contents of your cefdata flex table:

```
=> select maptostring(__raw__) from cefdata;  
maptostring
```

```
-----  
{  
  "deviceproduct" : "ArcSight",  
  "devicevendor" : "ArcSight",  
  "deviceversion" : "2.4.1",  
  "name" : "New alert",  
  "severity" : "High",  
  "signatureid" : "machine:20",  
  "version" : "0"  
}
```

```
(1 row)
```

4. Select some virtual columns from the `cefdata` flex table:

```
= select deviceproduct, severity, deviceversion from cefdata;
deviceproduct | severity | deviceversion
-----+-----+-----
ArcSight      | High    | 2.4.1
(1 row)
```

For more information, see [Loading Common Event Format \(CEF\) Data](#)

See Also

- [FAVROPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FJSONPARSER](#)
- [FREGEXPARSER](#)

FCSVPARSER

Parses CSV format (comma-separated values) data. Use this parser to load CSV data into columnar, flex, and hybrid tables. All data must be encoded in Unicode UTF-8 format. The parser `fcsvparser` supports the [RFC 4180](#) de facto standard for CSV data, and other options, to accommodate variations in CSV file format definitions. Invalid records will be rejected.

The `fcsvparser` does not support multibyte data. For more information about data formats, see [Checking Data Format Before or After Loading](#).

Parameters

<code>type= {'rfc4180' 'traditional' }</code>	CHAR	<p>[Optional] Specifies the default parameter values for the parser. You do not have to use the type parameter when loading data that conforms to the RFC 4180 standard (such as MS Excel files). See Loading CSV Data for the RFC4180 default parameters, and other options you can specify for traditional CSV files.</p> <p>Default value: RFC4180</p>
<code>delimiter</code>	CHAR	<p>[Optional] Indicates the single-character value used to separate fields in the CSV data.</p> <p>Default value: , (for rfc4180 and traditional)</p>
<code>escape</code>	CHAR	<p>[Optional] Specifies a single-character value. Use an escape character to interpret the next character in the data literally.</p> <p>Default value: " (for rfc4180) Default value: \ (for traditional)</p>
<code>enclosed_by</code>	CHAR	<p>[Optional] Specifies a single-character value. Use and enclosed_by value to include a value that is identical to the delimiter, but should be interpreted literally. For example, if the data delimiter is a comma (,), and you want to use a comma within the data ("my name is jane, and his is jim").</p> <p>Default value: " (for rfc4180 and traditional)</p>
<code>record_terminator</code>	CHAR	<p>[Optional] Indicates the single-character value used to specify the end of a record.</p>

		Default value: <code>\n</code> or <code>\r\n</code> (for rfc4180 and traditional)
header	BOOLEAN	<p>[Optional] Specifies whether to use the first row of data as a header column. When <code>header=true</code> (default), and no header exists, <code>fcsvparser</code> uses a default column heading. The default header consists of <code>ucoln</code>, where <code>n</code> is the column offset number, starting with <code>0</code> for the first column. You can specify custom column heading names using the <code>header_names</code> parameter, described next.</p> <p>If you specify <code>header=false</code>, the <code>fcsvparser</code> parses the first row of input as data, rather than as column headers.</p> <p>Default value: <code>true</code></p>
header_names	CHAR	<p>[Optional] Specifies a list of column header names, delimited by the character defined by the parser's <code>delimiter</code> parameter. Use this parameter to specify header names in a CSV file without a header row, or to override the column names present in the CSV source. To override one or more existing column names, specify the header names to use. This parameter overrides any header row in the data.</p>
trim	BOOLEAN	<p>[Optional] Indicates whether to trim white space from header names and key values.</p> <p>Default value: <code>true</code></p>
omit_empty_keys	BOOLEAN	<p>[Optional] Indicates how the parser handles header keys without values. If <code>omit_empty_keys=true</code>, keys with an empty value in the header row are not loaded.</p> <p>Default value: <code>false</code></p>

<code>reject_on_duplicate</code>	BOOLEAN	[Optional] Specifies whether to ignore duplicate records (<code>false</code>), or to reject duplicates (<code>true</code>). In either case, the load continues. Default value: <code>false</code>
<code>reject_on_empty_key</code>	BOOLEAN	[Optional] Specifies whether to reject any row containing a key without a value. Default value: <code>false</code>
<code>reject_on_materialized_type_error</code>	BOOLEAN	[Optional] Indicates whether to reject any materialized column value that the parser cannot coerce into a compatible data type. See Loading CSV Data . Default value: <code>false</code>

Examples

This example shows how you can use `fcsvparser` to load a flex table, build a view, and then query that view.

1. Create a flex table for CSV data:

```
=> CREATE FLEX TABLE rfc();  
CREATE TABLE
```

2. Use `fcsvparser` to load the data from STDIN. Specify that no header exists, and enter some data as shown:

```
=> COPY rfc FROM stdin PARSER fcsvparser(header='false');  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 10,10,20  
>> 10,"10",30  
>> 10,"20""5",90  
>> \.
```

3. Run the `compute_flextable_keys_and_build_view` function, and query the `rfc_view`. Notice that the default `enclosed_by` character permits an escape character (`"`)

within a field ("20" "5"). Thus, the resulting value was parsed correctly. Since no header existed in the input data, the function added `ucoln` for each column:

```
=> SELECT compute_flextable_keys_and_build_view('rfc');
           compute_flextable_keys_and_build_view
-----
Please see public.rfc_keys for updated keys
The view public.rfc_view is ready for querying
(1 row)

=> SELECT * FROM rfc_view;
 ucol0 | ucol1 | ucol2
-----+-----+-----
  10   |  10   |  20
  10   |  10   |  30
  10   | 20"5  |  90
(3 rows)
```

For more information and examples of using other parameters of this parser, see [Loading CSV Data](#).

See Also

- [FAVROPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FDELIMITEDPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FJSONPARSER](#)
- [FREGEXPARSER](#)

FDELIMITEDPAIRPARSER

Parses delimited data files. This parser provides a subset of the functionality in the parser `fdelimitedparser`. Use the `fdelimitedpairparser` when the data you are loading specifies pairs of column names with data in each row.

Parameters

delimiter	CHAR	[Optional] Specifies a single-character delimiter. Default value: ' '
record_terminator	CHAR	[Optional] Specifies a single-character record terminator. Default value: newline
trim	BOOLEAN	[Optional] Trims white space from header names and key values. Default value: true

Examples

The following example illustrates creating a sample flex table for simple delimited data, with two real columns, `eventId` and `priority`.

1. Create a table:

```
=> create flex table CEFData(eventId int default(eventId::int), priority int default
(priority::int) );
CREATE TABLE
```

2. Load a sample delimited Micro Focus ArcSight log file into the CEFData table, using the `fcefparser`:

```
=> copy CEFData from '/home/release/kmm/flextables/sampleArcSight.txt' parser
fdelimitedpairparser();
Rows Loaded | 200
```

3. After loading the sample data file, use `maptostring()` to display the virtual columns in the `__raw__` column of CEFData:

```
=> select maptostring(__raw__) from CEFData limit 1;
```

maptostring

```
-----  
"agentassetid" : "4-WwHuD0BABCCQDVaeX21vg==",  
"agentzone" : "3083",  
"agt" : "265723237",  
"ahost" : "svsvm0176",  
"aid" : "3tGoHuD0BABCCMDVaeX21vg==",  
"art" : "1099267576901",  
"assetcriticality" : "0",  
"at" : "snort_db",  
"atz" : "America/Los_Angeles",  
"av" : "5.3.0.19524.0",  
"cat" : "attempted-recon",  
"categorybehavior" : "/Communicate/Query",  
"categorydevicegroup" : "/IDS/Network",  
"categoryobject" : "/Host",  
"categoryoutcome" : "/Attempt",  
"categorysignificance" : "/Recon",  
"categorytechnique" : "/Scan",  
"categorytupledescription" : "An IDS observed a scan of a host.",  
"cnt" : "1",  
"cs2" : "3",  
"destinationgeocountrycode" : "US",  
"destinationgeolocationinfo" : "Richardson",  
"destinationgeopostalcode" : "75082",  
"destinationgeoregioncode" : "TX",  
"destinationzone" : "3133",  
"device product" : "Snort",  
"device vendor" : "Snort",  
"device version" : "1.8",  
"deviceseverity" : "2",  
"dhost" : "198.198.121.200",  
"dlat" : "329913940429",  
"dlong" : "-966644973754",  
"dst" : "3334896072",  
"dtz" : "America/Los_Angeles",  
"dvchost" : "unknown:eth1",  
"end" : "1364676323451",  
"eventid" : "1219383333",  
"fdevice product" : "Snort",  
"fdevice vendor" : "Snort",  
"fdevice version" : "1.8",  
"fdtz" : "America/Los_Angeles",  
"fdvchost" : "unknown:eth1",  
"lblstring2label" : "sig_rev",  
"locality" : "0",  
"modelconfidence" : "0",  
"mrt" : "1364675789222",  
"name" : "ICMP PING NMAP",  
"oagentassetid" : "4-WwHuD0BABCCQDVaeX21vg==",  
"oagentzone" : "3083",  
"oagt" : "265723237",  
"oahost" : "svsvm0176",  
"oaid" : "3tGoHuD0BABCCMDVaeX21vg==",  
"oat" : "snort_db",  
"oatz" : "America/Los_Angeles",
```

```
"oav" : "5.3.0.19524.0",  
"originator" : "0",  
"priority" : "8",  
"proto" : "ICMP",  
"relevance" : "10",  
"rt" : "1099267573000",  
"severity" : "8",  
"shost" : "198.198.104.10",  
"signature id" : "[1:469]",  
"slat" : "329913940429",  
"slong" : "-966644973754",  
"sourcegeocountrycode" : "US",  
"sourcegeolocationinfo" : "Richardson",  
"sourcegeopostalcode" : "75082",  
"sourcegeoregioncode" : "TX",  
"sourcezone" : "3133",  
"src" : "3334891530",  
"start" : "1364676323451",  
"type" : "0"  
}
```

(1 row)

4. Select the eventID and priority real columns, along with two virtual columns, atz and destinationgeoregioncode:

```
=> select eventID, priority, atz, destinationgeoregioncode from CEFData limit 10;  
eventID | priority | atz | destinationgeoregioncode  
-----+-----+-----+-----  
1218325417 | 5 | America/Los_Angeles |  
1219383333 | 8 | America/Los_Angeles | TX  
1219533691 | 9 | America/Los_Angeles | TX  
1220034458 | 5 | America/Los_Angeles | TX  
1220034578 | 9 | America/Los_Angeles |  
1220067119 | 5 | America/Los_Angeles | TX  
1220106960 | 5 | America/Los_Angeles | TX  
1220142122 | 5 | America/Los_Angeles | TX  
1220312009 | 5 | America/Los_Angeles | TX  
1220321355 | 5 | America/Los_Angeles | CA  
(10 rows)
```

See Also

- [FAVROPARSER](#)
- [FCEFPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPARSER](#)

- [FJSONPARSER](#)
- [FREGEXPARSER](#)

FDELIMITEDPARSER

Parses data using a delimiter character to separate values. The `fdelimitedparser` loads delimited data, storing it in a single-value VMap. You can use this parser to load data into columnar and flex tables.

Note: By default, `fdelimitedparser` treats empty fields as NULL, rather than as an empty string (' '). This behavior makes casting easier. Casting a NULL to an integer (`NULL::int`) is valid, while casting an empty string to an integer (`' '::int`) is not. If required, use the `treat_empty_val_as_null` parameter to change the default behavior of `fdelimitedparser`.

Parameters

<code>delimiter</code>	CHAR	[Optional] Indicates a single-character delimiter. Default value:
<code>record_terminator</code>	CHAR	[Optional] Indicates a single-character record terminator. Default value: \n
<code>trim</code>	BOOLEAN	[Optional] Determines whether to trim white space from header names and key values. Default value: true
<code>header</code>	BOOLEAN	[Optional] Specifies that a header column exists. The parser uses <code>col###</code> for the column names if you use this parameter but no header exists. Default value: true

<code>omit_empty_keys</code>	BOOLEAN	[Optional] Indicates how the parser handles header keys without values. If <code>omit_empty_keys=true</code> , keys with an empty value in the header row are not loaded. Default value: <code>false</code>
<code>reject_on_duplicate</code>	BOOLEAN	[Optional] Specifies whether to ignore duplicate records (<code>false</code>), or to reject duplicates (<code>true</code>). In either case, the load continues. Default value: <code>false</code>
<code>reject_on_empty_key</code>	BOOLEAN	[Optional] Specifies whether to reject any row containing a key without a value. Default value: <code>false</code>
<code>reject_on_materialized_type_error</code>	BOOLEAN	[Optional] Indicates whether to reject any row value for a materialized column that the parser cannot coerce into a compatible data type. See Using Flex Table Parsers . Default value: <code>false</code>
<code>treat_empty_val_as_null</code>	BOOLEAN	[Optional] Specifies that empty fields become NULLs, rather than empty strings (<code>' '</code>). Default value: <code>true</code>

Examples

1. Create a flex table for delimited data:

```
t=> CREATE FLEX TABLE delim_flex ();  
CREATE TABLE
```

2. Use the `fdelimitedparser` to load some delimited data from STDIN, specifying a comma (,) column delimiter:

```
=> COPY delim_flex FROM STDIN parser fdelimitedparser (delimiter=',');
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> deviceproduct, severity, deviceversion
>> ArcSight, High, 2.4.1
>> \.
```

You can now query virtual columns in the `delim_flex` flex table:

```
=> SELECT deviceproduct, severity, deviceversion from delim_flex;
deviceproduct | severity | deviceversion
-----+-----+-----
ArcSight      | High    | 2.4.1
(1 row)
```

See Also

- [FAVROPARSER](#)
- [FCEFPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FJSONPARSER](#)
- [FREGEXPARSER](#)

FJSONPARSER

Parses and loads a JSON file. This file can contain either repeated JSON data objects (including nested maps), or an outer list of JSON elements. For a flex table, the parser stores the JSON data in a single-value VMap. For a hybrid or columnar table, the parser loads data directly in any table column with a column name that matches a key in the JSON source data.

Parameters

<code>flatten_maps</code>	BOOLEAN	[Optional] Flattens sub-maps within the JSON data, separating map levels with a period (.).
---------------------------	---------	---

		Default value: true
flatten_arrays	BOOLEAN	[Optional] Converts lists to sub-maps with integer keys. Lists are not flattened by default. Default value: false
reject_on_duplicate	BOOLEAN	[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues. Default value: false
reject_on_empty_key	BOOLEAN	[Optional] Rejects any row containing a key without a value (reject_on_empty_key=true). Default value: false
omit_empty_keys	BOOLEAN	[Optional] Omits any key from the load data that does not have a value (omit_empty_keys=true). Default value: false
record_terminator	STRING	[Optional] When set, any invalid JSON records are skipped and parsing continues with the next record. Records must be terminated uniformly. For example, if your input file has JSON records terminated by newline characters, specify <code>fjsonparser(record_terminator=E'\n')</code> . If any invalid JSON records exist, parsing continues after the next <code>record_terminator</code> . When you do not use a <code>record_terminator</code> , parsing ends at the first invalid JSON record. Default value: no default value
reject_on_materialized_type_error	BOOLEAN	[Optional] Rejects a data row that contains a materialized column value that cannot be coerced into a compatible data type (<code>reject_on_materialized_type_error=true</code>). Default value: false
start_point	CHAR	[Optional] Specifies the name of a key in the

		<p>JSON load data at which to begin parsing. The parser ignores all data before the <code>start_point</code> value. The parser processes data after the first instance, and up to the second, ignoring any remaining data.</p> <p>Default value: none</p>
<code>start_point_occurrence</code>	INTEGER	<p>[Optional] Indicates the <i>n</i>th occurrence of the value you specify with <code>start_point</code>. Use in conjunction with <code>start_point</code> when load data has multiple start values and you know the occurrence at which to begin parsing.</p> <p>Default value: 1</p>
<code>suppress_nonalphanumeric_key_chars</code>	BOOLEAN	<p>[Optional] Suppresses non-alphanumeric characters in JSON key values. The parser replaces these characters with an underscore (<code>_</code>) when this parameter is <code>true</code>.</p> <p>Default value: false</p>
<code>key_separator</code>	CHAR	<p>[Optional] Specifies a non-default character for the parser to use when concatenating key names.</p> <p>Default value: ' . '</p>

Examples

Load JSON Data Without Optional Parameters

1. Create a flex table, `super`, with two columns, `age` and `name`:

```
=> create table super(age int, name varchar);  
CREATE TABLE
```

2. Enter values using the `fjsonparser()`, and query the results:

```
=> copy super from stdin parser fjsonparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"age": 5, "name": "Tim"}  
>> {"age": 3}
```

```
>> {"name": "Fred"}
>> {"name": "Bob", "age": 10}
>> \.
=> select * from super;
  age | name
-----+-----
      | Fred
  10  | Bob
   5  | Tim
   3  |
(4 rows)
```

For other examples, see [Loading JSON Data](#).

See Also

- [FAVROPARSER](#)
- [FCEFPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPARSER](#)
- [FDELIMITEDPAIRPARSER](#)
- [FREGEXPARSER](#)

FREGEXPARSER

Parses a regular expression, matching columns to the contents of the named regular expression groups.

Parameters

pattern	VARCHAR	Specifies the regular expression of data to match. Default value: Empty string ("")
use_jit	BOOLEAN	[Optional] Indicates whether to use just-in-time compiling when parsing the regular expression.

		Default value: false
record_terminator	VARCHAR	[Optional] Specifies the character used to separate input records. Default value: \n
logline_column	VARCHAR	[Optional] Captures the destination column containing the full string that the regular expression matched. Default value: Empty string ("")

Example

These examples use the following regular expression, which searches for information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.

This example expression loads any thread_id hex value, regardless of whether it has a 0x prefix, (<thread_id>(?:0x)?[0-9a-f]+).

```
'^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)
(?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
-?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
\<(?!<level>\w+)\> )?(?:<(?!<level>\w+)\> @[(?!<enode>\w+)]?: )
?(?!<text>.*).'
```

1. Create a flex table (vlog) to contain the results of a Vertica log file. For this example, we made a copy of a log file in the directory /home/dbadmin/data/vertica.log:

```
=> create flex table vlog1();
CREATE TABLE
```

2. Use the fregexparser with the sample regular expression to load data from the log file. Be sure to remove any line characters before using this expression shown here:

```
=> copy vlog1 from '/home/dbadmin/tempdat/KMvertica.log'
PARSER FREGEXPARSER('^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)
(?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
-?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
\<(?!<level>\w+)\> )?(?:<(?!<level>\w+)\> @[(?!<enode>\w+)]?: )
?(?!<text>.*)');
```

```
Rows Loaded
-----
      31049
(1 row)
```

3. After successfully loading data, use the `MAPTOSTRING()` function with the table's `__raw__` column. The four rows (`limit 4`) that the query returns are regular expression results of the `KMvertica.log` file, parsed with `fregexparser`. The output shows `thread_id` values with a preceding `0x` or without:

4.

```
=> select maptostring(__raw__) from vlog1 limit 4;
      maptostring
-----
{
  "text" : " [Init] <INFO> Log /home/dbadmin/VMart/v_vmart_node0001_catalog/vertica.log
opened; #2",
  "thread_id" : "0x7f2157e287c0",
  "thread_name" : "Main",
  "time" : "2017-03-21 23:30:01.704"
}

{
  "text" : " [Init] <INFO> Processing command line: /opt/vertica/bin/vertica -D
/home/dbadmin/VMart/v_vmart_node0001_catalog -C VMart -n v_vmart_node0001 -h
10.20.100.247 -p 5433 -P 4803 -Y ipv4",
  "thread_id" : "0x7f2157e287c0",
  "thread_name" : "Main",
  "time" : "2017-03-21 23:30:01.704"
}

{
  "text" : " [Init] <INFO> Starting up Vertica Analytic Database v8.1.1-20170321",
  "thread_id" : "7f2157e287c0",
  "thread_name" : "Main",
  "time" : "2017-03-21 23:30:01.704"
}

{
  "text" : " [Init] <INFO> Compiler Version: 4.8.2 20140120 (Red Hat 4.8.2-15)",
  "thread_id" : "7f2157e287c0",
  "thread_name" : "Main",
  "time" : "2017-03-21 23:30:01.704"
}

(4 rows)
```

See Also

- [FDELIMITEDPAIRPARSER](#)
- [FDELIMITEDPARSER](#)
- [FJSONPARSER](#)

Using Management Console

Management Console (MC) is a user-friendly performance monitoring and management tool that provides a unified view of your Vertica database operations. Using a browser, you can create, import, manage, and monitor one or more databases and their associated clusters. You can also create and manage MC users. You can then map the MC users to a Vertica database and manage them through the MC interface.

The MC interface provides tool tips for most of the MC operations. For an introduction to MC functionality, architecture, and security, see [Management Console](#) in Vertica Concepts.

To get started using MC, see [Getting Started with MC](#).

Getting Started with MC

Use Management Console to monitor the performance of your Vertica clusters. This tool provides a graphical view of your Vertica database cluster, nodes, network status, and detailed monitoring charts and graphs.

MC allows you to:

- Create, import, and connect to Vertica database.
- Manage your Vertica database and clusters.
- Receive and view messages regarding the health and performance of your Vertica database and clusters.
- View diagnostics and support information for Management Console.
- Manage application and user settings for Management Console.

MC Installation Process

To install MC, complete these tasks:

1. Verify that you meet the requirements listed in [Before You Install MC](#).
2. Follow the steps listed in [Installing Management Console](#).
3. After you have installed MC, configure it according to the instructions in [Configuring MC](#).

Connecting to MC

To connect to Management Console:

1. Open an HTML-5 compliant browser.
2. Enter the IP address or host name of the host on which you installed MC (or any cluster node if you installed Vertica first), followed by the MC port you assigned when you [configured MC](#).

For example, enter one of:

```
https://00.00.00.00:5450/
```

or

```
https://hostname:5450/
```

3. When the MC logon dialog appears, enter your MC username and password and click **Log in**.

Note: When MC users log in to the MC interface, MC checks their privileges on Vertica Data Collector (DC) tables on MC-monitored databases. Based on DC table privileges, along with the role assigned the MC user, each user's access to the MC's Overview, Activity and Node details pages could be limited. See [About MC Privileges and Roles](#) for more information.

If you do not have an MC username/password, contact your MC administrator.

Viewing the Home Page

After you [connect to MC](#) and sign in, the Home page displays. This page is the entry point to all Vertica database clusters and users managed by MC. Information on this page, as well as throughout the MC interface, will appear or be hidden, based on the permissions ([access levels](#)) of the user who is logged in. The following image is what an MC super administrator sees.

Prepare data and databases



Provisioning
Create and import databases and clusters

Manage Information



Infrastructure
Manage databases and clusters



Message Center
Go to the database message center



Diagnostics
View diagnostics and support information



MC Settings
Manage application and user settings

Recent Databases



doccdb



VMart_BR

Tasks

Operations you can perform in Management Console are grouped into the following task-based areas:

- **Provision Databases.** Create and import databases, and connect this includes creating new empty databases and importing existing database clusters into Management Console. You can also set up an Ambari connection to import a Vertica cluster that resides in a Hadoop environment. See [Managing Database Clusters](#).
- **Existing Infrastructure.** View all the clusters and databases monitored by MC, stop and remove databases, and view details about databases and clusters. See [Managing Database Clusters](#).
- **MC Settings.** Configure MC and user settings, as well as use the MC interface to install Vertica on a cluster of hosts. See [Managing MC Settings](#).
- **Message Center.** View, sort, and search database messages and optionally export messages to a file. See [Monitoring Database Messages in MC](#).
- **Diagnostics.** View and resolve MC-related issues, as well as browse Vertica agent and audit logs. See [Troubleshooting with MC Diagnostics](#).

Recent Databases

The **Recent Databases** section displays all databases that you created on or imported into the MC interface. You can install one or more databases, on the same cluster or on different clusters, but you can have only one database running on a single cluster at a time. UP databases appear in green and DOWN databases are red. An empty space under Recent Databases means that you have not yet created or imported a database into the MC interface, or do not have permission to view any databases managed by MC.

Managing MC Users, Roles and Privileges

If you are an administrator, you can use **MC Settings** to grant MC users privileges to one or more Vertica users. MC users are not the same as system (Linux) users. MC users are external to the database, and their information is stored on an internal database on the MC application or web server. See [About MC Users](#) for further details.

You can create MC users using either of two authentication techniques, **LDAP** or **MC (internal)**. See [Creating an MC User](#). After you create the MC users, you can manage them from **MC Settings** page. Refer to [Managing MC Users](#).

To control the level of access for the MC Users, you can grant them privileges (through roles) from the **MC Settings** page. MC supports two groups of privileges:

- [MC Configuration Privileges](#)
- [MC Database Privileges](#)

The MC super account is the default user. The super user needs to create all other MC users. Refer to [About MC Privileges and Roles](#) for further information on MC roles.

For further details about MC Users, Privileges and Roles, see [Managing Users And Privileges](#).

Creating a Cluster Using MC - Process Flow

After you install and configure MC, you can use it to create a Vertica cluster on hosts where Vertica software is not installed. Complete the following tasks:

1. [Prepare the Hosts](#) - Prepare each host that will become a node in the cluster.
2. [Create a Private Key File](#) - MC needs password-less SSH to connect to hosts and install Vertica software. Create a private key to enable MC access to the hosts.
3. [Use the MC Cluster Installation Wizard](#) - Use the wizard to install a Vertica cluster on hosts that do not have Vertica software already installed on them.
4. [Validate Hosts and Create the Cluster](#) - Host validation is the process where the MC runs tests against each host in a [proposed cluster](#). You must validate hosts before the MC can install Vertica on each host.

After you successfully create a cluster using MC, see [Create a Database on a Cluster](#).

Creating a Database Using MC

Follow these steps to create a database on an existing cluster using Management Console.

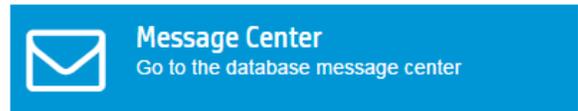
Important: This task assumes that you have already installed Management Console and are familiar with its concepts and layout. For more information, see [Using Management Console](#), [Management Console](#) in the Concepts Guide, and [Installing and Configuring Management Console](#) in the Installation Guide.

1. Connect to Management Console, and log in.
2. On the Home page, under the **Manage Information** pane, click **Existing Infrastructure**.

Prepare data and databases

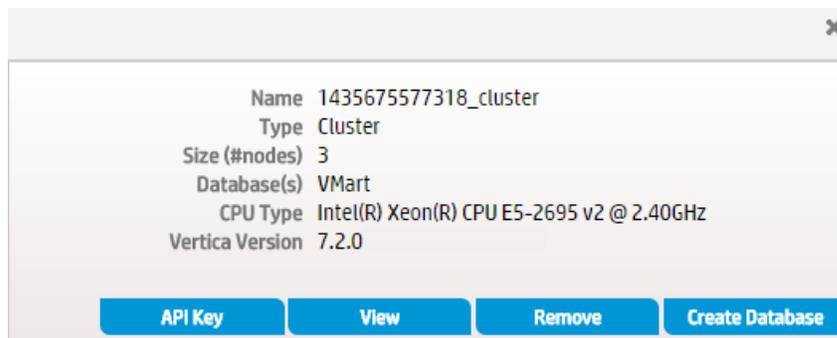


Manage Information



Recent Databases

3. On the following screen, click to select an existing cluster. When a dialog box identifying that cluster appears, click **Create Database**. The database creation wizard starts.



4. Follow the steps in the wizard to successfully create a database.

See Also

- [Create an Empty Database Using MC](#)
- [Import an Existing Database Into MC](#)
- [Creating a Cluster Using MC - Process Flow](#)

Provisioning Databases Using MC

Management Console allows all users to create, import, and connect to Vertica databases using the **MC Provision Databases** tab.

- Import cluster or database using IP discovery
- Create a new cluster
- [Import and Monitor in a Hadoop Environment](#)

Monitoring Existing Infrastructure Using MC

Use the **Management Console Existing Infrastructure** tab to monitor the health of your Vertica databases and clusters. Click the cluster of interest to view the health of the nodes in that cluster and the key information associated with the cluster such as:

- Vertica version
- Number of hosts
- CPU type
- Last updated date
- Node list.

You can also zoom in and out for better view of this page.

Click the database which you want to monitor to go to its Overview page:



Tip: You can also view the **Overview** page by clicking on the desired database on the **MC Home Page**.

You can perform the following tasks from the Overview page:

- View **Quick Stats** to get instant alerts and information about your cluster's status.
- View **Status Summary** that provides a general overview of the status of your cluster (as shown in preceding figure).
- Analyze **System Health** using a comprehensive summary of your system resource usage and node information, with configurable statistics that allow you to specify acceptable ranges of resource usage.
- Use **Query Synopsis** to monitor system query activity and resource pool usage.

Additionally, you can perform the following tasks from the Overview page:

- [Monitoring Cluster Nodes](#)
- [Monitoring Cluster CPU/Memory](#)
- [Monitoring Cluster Performance](#)

Monitoring System Resources

On the main window, you can click the database, and navigate to the **MC Activity** tab to monitor system resources such as:

- [Queries](#)
- [Internal Sessions](#)
- [User Sessions](#)
- [Memory Usage](#)
- [System Bottlenecks](#)
- [User Query Phases](#)
- [Table Treemap](#)
- [Query Monitoring](#)
- [Resource Pool Monitoring](#)

Monitoring Node and MC User Activity

You can use the **MC Manage** page to monitor node activity. When you click the node you want to investigate, the Node Detail page opens and provides:

- Summary information for the node
- Resources consumed by the node for last three hours

You can also browse and export log-level data from AgentTools and Vertica log files. MC retains a maximum of 2000 log records. See [Monitoring Node Activity](#) for further details.

Use **MC Diagnostics** tab and navigate to **Audit Log** page to manage MC User activity. See [Monitoring MC User Activity Using Audit Log](#).

Monitoring messages in Databases managed by MC

You can view critical database related messages from **MC Message Center**. The **MC Message Center** reports on several critical database-related conditions using a color code to indicate the message severity. See [Monitoring Database Messages in MC](#) for further details.

You can also [search](#) and sort database messages, mark messages read or unread and delete them. You can filter messages by message type, and [export](#) messages. Refer to [Searching Database Messages Managed by MC](#) and [Exporting MC-managed Database Messages and Logs](#).

Monitoring and Configuring Resource Pools

Use the **MC Activity** page to monitor resource pools. Select the resource pool you want to monitor. MC displays the following charts for the selected pool:

- Resource Usages in Pool
- Memory Usage in Node
- Average Query Execution and Query Time in Pool
- Resource Rejections in Pool

If you are a database administrator, you can click the database you want on the main window. You can then use the **MC Settings** tab to view and edit the resource pool parameters. Only the database administrator can monitor and configure the resource pools in Management Console. See [Monitoring Resource Pools](#) for further information.

Running Database Designer Using MC

You can use Database Designer to create a comprehensive design, which allows you to create new projections for all tables in your database.

Additionally, you can use Database Designer to create an incremental design. An incremental design creates projections for all tables referenced in the queries you supply.

To run Database Designer using MC, follow the steps listed at [Running Database Designer with Management Console](#) .

Managing Queries Using MC

Management Console allows you to view the query plan of an active query or a manually entered query specified by the user.

1. On the **MC Home Page**, click the database you want to view the **Overview** page.
2. Select the **Activity** tab to view the query activity.
3. Click the **Explain** tab to access the query plan.

See [Managing Queries in MC](#) and [Accessing Query Plans in Management Console](#) for further information.

Management Console provides two options for viewing the query plan: **Path Information** and **Tree Path**. For details on each, refer [Query Plan View Options](#).

Additionally, you can also [Viewing Projection and Column Metadata](#) using the **MC Explain** tab.

See Also

- [Expanding and Collapsing Query Paths](#)
- [Clearing Query Data](#)

Profiling Queries Using MC

Management Console allows you to view profile data for a query.

- On the **MC Home Page**, click the database to view the **Overview** page.
- Click the **Explain** tab to perform tasks related to profiling a query.

See [Viewing Profile Data in MC](#) for further details.

On the **Explain** tab, you can view the following profile data using MC:

- [Query Phase Duration](#)
- [Projection Metadata](#)

- [Execution Events](#)
- [Optimizer Events](#)
- [Profile Metrics](#)

You can use any of the four different formats to view the profile data:

- Path Information view
- Query Drilldown view
- Tree Path view
- Profile Analysis view

See [Viewing Different Profile Outputs](#) for detailed explanation of each view.

Additionally, Management Console supports different color codes for viewing the progress of profiling a query. For an explanation of these color codes, see [Monitoring Profiling Progress](#).

See Also

- [Expanding and Collapsing Query Path Profile Data](#)
- [Clearing Profile Data](#)

Managing Client Connections

Each client session to MC uses a connection from `MaxClientSessions`, a database configuration parameter. This parameter determines the maximum number of sessions that can run on a single database cluster node. Sometimes multiple MC users, mapped to the same database account, are concurrently monitoring the Overview and Activity pages. In such cases, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for `MaxClientSessions` on an MC-monitored database to account for extra sessions. See [Managing Sessions](#) for details.

Managing MC Settings

The **MC Settings** page allows you to configure properties specific to Management Console. You can:

- Change MC and agent default port assignments.
- Enable and disable username and password auto-complete at MC login. (After disabling, clear your browser's cache.)
- Control the following monitoring settings in MC:
 - Enable checks and set alert thresholds for spread retransmit rate. This setting is disabled by default. The recommended alert threshold for spread retransmit rate is 10%.
 - Set alert thresholds for free MC disk space checks. The recommended alert threshold is 500 MB.
 - Exclude MC queries from activity charts.
 - Set refresh intervals for MC charts and pages.
- Upload a new SSL certificate.
- Use LDAP for user authentication.
- Create new MC users and, with their user credentials, map them to a database managed by MC on the Vertica server. See [Creating an MC User](#) and [Managing MC Users](#).
- View your version of Vertica or upload a new Vertica binary file.
- Customize the look and feel of MC with themes. See [Customizing Look and Feel](#).
- Configure MC to use an alternative data source to monitor your database. See [Monitoring External Data Sources in Management Console](#).
- Enable MC to send email alerts. See [Set Up Email](#).
- Configure [Extended Monitoring](#), which allows you to monitor more long-term data in MC:
 - Set up an external storage database for Extended Monitoring. See [Managing the Storage Database](#).

- Enable or disable Extended Monitoring on your databases. See [Managing Extended Monitoring on a Database](#).

Modifying Database-Specific Settings

To inspect or modify settings related to a database managed by MC, go to the **Existing Infrastructure** page. On this page, select a running database to see its Overview page. From the bottom of the Overview page, click the Settings tab to make modifications to database-specific settings.

Backing Up MC

Before you [upgrade MC](#), Micro Focus recommends that you back up your MC metadata (configuration and user settings). Use a storage location external to the server on which you installed MC.

1. On the target server (where you want to store MC metadata), log in as root or a user with sudo privileges.
2. Create a backup directory as in following example:

```
# mkdir /backups/mc/mc-backup-20130425
```

3. Copy the `/opt/vconsole` directory to the new backup folder:

```
# cp -r /opt/vconsole /backups/mc/mc-backup-20130425
```

Upgrading And Uninstalling MC

To upgrade or uninstall MC refer to [Installing Vertica](#)

- To upgrade MC, follow the steps listed on [Upgrading Management Console](#).
- To uninstall MC, refer [Uninstalling Management Console](#).

Managing Users And Privileges

A Management Console administrator can grant MC users access to one or more Vertica databases through the MC interface. In this section, we discuss about MC Users and their privileges.

- [About MC Users](#)
- [About MC Privileges and Roles](#)

About MC Users

Unlike database users, which you create on the Vertica database and then grant privileges and roles through SQL statements, you create MC users on the Management Console interface. MC users are external to the database. Their information is stored on an internal database on the MC application/web server. Their access to both MC and to databases managed by MC is controlled by groups of privileges (also referred to as access levels). MC users are not system (Linux) users; they are entries in the MC internal database.

Permission Group Types

There are two types of permission groups on MC, those that apply to MC configuration and those that apply to database access:

- [MC configuration](#) privileges are made up of roles that control what users can configure on the MC, such as modify MC settings, create and import Vertica databases, restart MC, create a Vertica cluster through the MC interface, and create and manage MC users.
- [MC database](#) privileges are made up of roles that control what users can see or do on a Vertica database monitored by MC, such as view the database cluster state, query and session activity, monitor database messages and read log files, replace cluster nodes, and stop databases.

If you are using MC, you might want to allow one or more users in your organization to configure and manage MC, and you might want other users to have database access only. You can meet these requirements by creating MC users and granting them a role from each privileges group. See [Creating an MC User](#) for details.

MC User Types

There are five types of role-based users on MC:

- The default superuser administrator (Linux account) who gets created when you install and configure MC and oversees all of MC. See [SUPER Role \(mc\)](#).
- Users who can configure all aspects of MC and control all databases managed by MC. See [ADMIN Role \(mc\)](#).

- Users who can configure MC user settings and monitor all databases managed by MC. See [MANAGER Role \(MC\)](#).
- Users who can configure some aspects of MC user settings and monitor all databases managed by MC. See [IT Role \(mc\)](#).
- Users who cannot configure MC and have access to one or more databases managed by MC. See [NONE Role \(mc\)](#).

You create users and grant them privileges (through roles) on the **MC Settings** page in the **User management** tab.

Creating Users and Choosing an Authentication Method

You create users and grant them privileges (through roles) on the **MC Settings** page. You can also choose how to authenticate their access to MC.

- To add users who are authenticated against the MC, click **User Management**
- To add users who are authenticated through your organization's LDAP repository, click **Authentication**

MC supports only one method for authentication, so if you choose MC, all MC users will be authenticated using their MC login credentials.

Default MC Users

The MC super account is the only default user. The super or another MC administrator must create all other MC users.

See Also

- [Management Console](#)
- [About MC Privileges and Roles](#)
- [Granting Database Access to MC Users](#)
- [Mapping an MC User to a Database user's Privileges](#)

Creating an MC User

MC provides two authentication schemes for MC users: LDAP or MC (internal). Which method you choose will be the method MC uses to authenticate *all* MC users. It is not possible to authenticate some MC users against LDAP and other MC users against credentials in the database through MC.

- **MC (internal) authentication.** Internal user authorization is specific to MC itself. You create a user with a username and password combination. This method stores MC user information in an internal database on the MC application/web server, and encrypts passwords. Note that these MC users are not system (Linux) users; they are entries in the MC's internal database.
- **LDAP authentication.** All MC users—except for the MC super administrator, which is a Linux account—are authenticated based on search criteria against your organization's LDAP repository. MC uses information from LDAP for authentication purposes only and does not modify LDAP information. Also, MC does not store LDAP passwords but passes them to the LDAP server for authentication.

Instructions for creating new MC users are in this topic.

- If you chose MC authentication, follow the instructions under **Create a New User Authenticated by MC**.
- If you chose LDAP authentication, follow the instructions under **Create a New User from LDAP**.

See [About MC Users](#) and [LDAP Authentication](#) for more information.

Prerequisites

Before you create an MC user, you already:

- Created a database directly on the server or through the MC interface, or you imported an existing database cluster into the MC interface. See [Managing Database Clusters](#).
- Created a database user account (source user) on the server, which has the privileges and/or roles you want to map to the new (target) MC user. See [Creating a Database User](#).

- Know what MC privileges you want to grant the new MC user. See [About MC Privileges and Roles](#).
- Are familiar with the concept of [mapping MC users to database users](#).

If you have not yet met the first two above prerequisites, you can still create new MC users; you just won't be able to map them to a database until after the database and target database user exist. To grant MC users database access later, see [Granting Database Access to MC Users](#).

Create a New User Authenticated by MC

1. Sign in to Management Console as an administrator and navigate to **MC Settings > User management**.
2. Click **Add**.
3. Enter the MC username.

Note: It is not necessary to give the MC user the exact same name as the database user account you'll map the MC user to in Step 7. What matters is that the source database user has privileges and/or roles similar to the database role you want to grant the MC user. The most likely scenario is that you will map multiple MC users to a single database user account. See [MC Database Privileges](#) and [Mapping an MC User to a Database user's Privileges](#) for more information.

4. Let MC generate a password or create one by clicking **Edit password**. If LDAP has been configured, the MC password field will not appear.
5. Optionally enter the user's e-mail address.
6. Select an **MC configuration permissions** level. See [MC Configuration Privileges](#).
7. Next to the **DB access levels** section, click **Add** to grant this user database permissions.
 - i. **Choose a database.** Select a database from the list of MC-discovered (databases that were created on or imported into the MC interface).
 - ii. **Database username.** Enter an existing database user name or, if the database is running, click the ellipses [...] to browse for a list of database users, and select a name from the list.

- iii. **Database password.** Enter the password to the database user account (not this username's password).
 - iv. **Restricted access.** Chose a database level ([ADMIN](#), [IT](#), or [USER](#)) for this user.
 - v. Click **OK** to close the **Add permissions** dialog box.
8. Leave the user's **Status** as enabled (the default). If you need to prevent this user from accessing MC, select disabled.
9. Click **Add User** to finish.

Create a New LDAP-authenticated User

When you add a user from LDAP on the MC interface, options on the **Add a new user** dialog box are slightly different from when you create users without LDAP authentication. Because passwords are store externally (LDAP server) the password field does not appear. An MC administrator can override the default LDAP search string if the user is found in another branch of the tree. The **Add user** field is pre-populated with the default search path entered when LDAP was configured.

1. Sign in to Management Console and navigate to **MC Settings > User management**.
2. Click **Add** and provide the following information:
 - a. LDAP user name.
 - b. LDAP search string.
 - c. User attribute, and click **Verify user**.
 - d. User's email address.
 - e. MC configuration role. NONE is the default. See [MC Configuration Privileges](#) for details.
 - f. Database access level. See [MC Database Privileges](#) for details.
 - g. Accept or change the default user's **Status** (enabled).
3. Click **Add user**.

If you encounter issues when creating new users from LDAP, you'll need to contact your organization's IT department.

How MC Validates New Users

After you click OK to close the Add permissions dialog box, MC tries to validate the database username and password entered against the selected MC-managed database or against your organization's LDAP directory. If the credentials are found to be invalid, you are asked to re-enter them.

If the database is not available at the time you create the new user, MC saves the username/password and prompts for validation when the user accesses the Database and Clusters page later.

See Also

- [Configuring MC](#)
- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [Granting Database Access to MC Users](#)
- [Creating a Database User](#)

Managing MC Users

You manage MC users through the following pages on the Management Console interface:

- **MC Settings > User management**
- **MC Settings > Resource access**

Who Manages Users

The MC superuser administrator ([SUPER Role \(mc\)](#)) and users granted [ADMIN Role \(mc\)](#) manage all aspects of users, including their access to MC and to MC-managed databases.

Users granted [IT Role \(mc\)](#) can enable and disable user accounts.

See [About MC Users](#) and [About MC Privileges and Roles](#) for more information.

Editing an MC user's information follows the same steps as [creating a new user](#), except the user's information will be pre-populated, which you then edit and save.

The only user account you cannot alter or remove from the MC interface is the MC super account.

What Kind of User Information You Can Manage

You can change the following user properties:

- MC password
- Email address. This field is optional. If the user is authenticated against LDAP, the email field is pre-populated with that user's email address if one exists.
- [MC Configuration Privileges](#) role
- [MC Database Privileges](#) role

You can also change a user's status (enable/disable access to MC) and delete users.

About User Names

After you create and save a user, you cannot change that user's MC user name, but you can delete the user account and create a new user account under a new name. The only thing you lose by deleting a user account is its audit activity, but MC immediately resumes logging activity under the user's new account.

About MC Privileges and Roles

As introduced in [About MC Users](#), you control user access to Management Console through groups of privileges (also referred to as access levels) that fall into two types, those that apply to MC configuration, and those that apply to MC-managed Vertica databases.

MC Permission Groups

- [MC configuration](#) privileges are made up of roles that control what users can configure on the MC, such as modify MC settings, create and import Vertica databases, restart MC, create a Vertica cluster through the MC interface, and create and manage MC users.
- [MC database](#) privileges are made up of roles that control what users can see or do on a Vertica database monitored by MC, such as view the database cluster state, query and session activity, monitor database messages and read log files, replace cluster nodes, and stop databases.

Note: When you grant an MC user a database role, that user inherits the privileges assigned to the database user account to which the MC user is mapped. For maximum access, use the dbadmin username and password.

MC database privileges cannot alter or override the Vertica database user's privileges and roles. MC user/database user association is described in [Mapping an MC User to a Database user's Privileges](#).

See Also

- [About MC Users](#)
- [Creating an MC User](#)
- [Managing MC Users, Roles and Privileges](#)
- [MC Database Privileges](#)
- [Creating an MC User](#)

- [Granting Database Access to MC Users](#)
- [Mapping an MC User to a Database user's Privileges](#)

MC Configuration Privileges

When you create an MC user, you assign them an MC configuration access level (role). MC roles control a user's ability to create users and manage MC settings on the MC interface.

In addition to an MC role, users also have a database role that controls their database-specific privileges. See [MC Database Privileges](#).

MC Roles and Privileges You Can Assign Users

You can assign a user one of the following MC access levels:

- [ADMIN Role \(mc\)](#)—Full access to all MC functionality.
- [MANAGER Role \(MC\)](#)—Access to MC user management functionality. Access to non-database MC alerts.
- [IT Role \(mc\)](#)—Limited access to MC user management functionality. Access to MC log and to non-database MC alerts.
- [NONE Role \(mc\)](#)—Database access only, to the databases an administrator assigns to this user.

You grant MC configuration privileges at the same time you create the user's account, on the User Management tab of the MC Settings page. You can change MC access levels using this page. See [Creating an MC User](#) for details.

You can also use the User Management tab to grant users access to one or more databases managed by MC. See [MC Database Privileges](#) for details.

MC Configuration Privileges By User Role

You grant the following configuration privileges by MC role.

MC access privileges	ADMIN	MANAGER	IT	NONE
Configure MC settings:	Yes			

MC access privileges	ADMIN	MANAGER	IT	NONE
<ul style="list-style-type: none"> Configure storage locations and ports Upload new SSL certificates Manage LDAP authentication Update Vertica installation Change MC theme Map to an external data source 				
Configure user settings: <ul style="list-style-type: none"> Add, edit, delete users Add, change, delete user permissions Map users to one or more databases 	Yes	Yes		
Configure user settings: <ul style="list-style-type: none"> Enable or disable user access to MC Reset user passwords 	Yes	Yes	Yes	
Monitor user activity on MC using audit log	Yes			
Create and manage databases and clusters: <ul style="list-style-type: none"> Create a new database or import an existing one Create a new cluster or import an existing one Remove databases and clusters from MC 	Yes			
Reset MC to its original, preconfigured state	Yes			
Restart Management Console	Yes			
View full list of databases monitored by MC	Yes	Yes	Yes	
View MC log	Yes		Yes	

MC access privileges	ADMIN	MANAGER	IT	NONE
View non-database MC alerts	Yes	Yes	Yes	Yes

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [Managing MC Users, Roles and Privileges](#)
- [MC Database Privileges](#)
- [Creating an MC User](#)
- [Granting Database Access to MC Users](#)
- [Mapping an MC User to a Database user's Privileges](#)

SUPER Role (mc)

The default superuser administrator, called **Super** on the MC UI, is a Linux user account that gets created when you install and [configure MC](#). During the configuration process, you can assign the Super any name you like; it need not be dbadmin.

The MC SUPER role, a superset of the [ADMIN Role \(mc\)](#), has the following privileges:

- Oversees the entire Management Console, including all MC-managed database clusters

Note: This user inherits the privileges/roles of the user name supplied when importing an Vertica database into MC. Micro Focus recommends that you use the database administrator's credentials.

- Creates the first MC user accounts and assigns them an MC configuration role
- Grants MC users access to one or more MC-managed Vertica databases by assigning [MC Database Privileges](#) to each user

The MC super administrator account is unique. Unlike other MC users you create, including other MC administrators, the MC super account cannot be altered or dropped, and you cannot grant the SUPER role to other MC users. The only property you can change for the MC super is

the password. Otherwise the SUPER role has the same privileges on MC as the [ADMIN Role \(mc\)](#).

On MC-managed Vertica databases, SUPER has the same privileges as [ADMIN Role \(db\)](#).

The MC super account does not exist within the LDAP server. This account is also different from the special dbadmin account that gets created during an Vertica installation, whose privileges are governed by the [DBADMIN Role](#). The Vertica-created dbadmin is a Linux account that owns the database catalog and storage locations and can bypass database authorization rules, such as creating or dropping schemas, roles, and users. The MC super does not have the same privileges as dbadmin.

See Also

- [Configuring MC](#)
- [About MC Privileges and Roles](#)
- [Creating an MC User](#)
- [Granting Database Access to MC Users](#)
- [Adding Multiple Users to MC-managed Databases](#)
- [Mapping an MC User to a Database user's Privileges](#)
- [Managing MC Users](#)

ADMIN Role (mc)

This user account is the user who can perform all administrative operations on Management Console, including configure and restart the MC process and add, change, and remove all user accounts. By default, MC administrators inherit the database privileges of the main database user account used to set up the database on the MC interface. Therefore, MC administrators have access to all MC-managed databases. Grant the ADMIN role to users you want to be MC administrators.

The difference between this ADMIN user and the default Linux account, the MC [SUPER role](#), is you cannot alter or delete the MC SUPER account, and you can't grant the SUPER role to any other MC users. You can, however, change the access level for other MC administrators, and you can delete this user's accounts from the MC interface.

The following list highlights privileges granted to the ADMIN role:

- Modify MC settings, such as storage locations and ports, restart the MC process, and reset MC to its original, unconfigured state
- Audit license activity and install/upgrade a Vertica license
- Upload a new SSL certificate
- Use LDAP for user authentication
- View the MC log, alerts and messages
- Add new users and map them to one or more Vertica databases by granting an [MC database-level role](#)
- Select a database and add multiple users at once
- Manage user roles and their access to MC
- Remove users from the MC
- Monitor user activity on the MC interface
- Stop and start any MC-managed database
- Create new databases/clusters and and import existing databases/clusters into MC
- Remove databases/clusters from the MC interface
- View all databases/clusters imported into MC

About the MC Database Administrator Role

There is also an MC database administrator (ADMIN) role that controls a user's access to MC-managed databases. The two ADMIN roles are similar, but they are not the same, and you do not need to grant users with the ADMIN (mc) role an ADMIN (db) role because MC ADMIN users automatically inherit all database privileges of the main database user account that was created on or imported into MC.

The following table summarizes the primary difference between the two ADMIN roles, but see [ADMIN Role \(db\)](#) for details specific to MC-managed database administrators.

MC configuration ADMIN role	MC database ADMIN role
Perform all administrative operations on the MC itself, including restarting the MC process.	Perform database-specific activities, such as stop and start the database, and

MC configuration ADMIN role	MC database ADMIN role
Privileges extend to monitoring all MC-created and imported databases but anything database-related beyond that scope depends on the user's privileges granted on the database through GRANT statements.	monitor query and user activity and resources. Other database operations depend on that user's privileges on the specific database. This ADMIN role cannot configure MC.

MANAGER Role (MC)

Users assigned the Manager role can configure user settings in MC. The Manager role allows full access to the User Management tab in MC Settings. Managers can also view a full list of databases monitored by MC on the Home page, view the MC log, and see non-database MC alerts.

The Manager role has similar configuration privileges to the IT configuration role. Unlike IT users, Managers can also create, edit, and delete users in User Settings.

Managers can:

- Add, edit, delete users
- Add, change, delete user permissions
- Map users to one or more databases
- Enable or disable user access to MC
- Reset user passwords
- View the full list of databases monitored by MC on the MC Home page
- View the MC log
- View non-database MC alerts

IT Role (mc)

MC IT users can monitor all MC-managed databases, view MC-level (non database) messages, logs, and alerts, disable or enable user access to MC, and reset non-LDAP user passwords. You can also assign MC IT users specific database privileges, which you do by mapping IT users to a user on a database. In this way, the MC IT user inherits the privileges assigned to the database user to which he/she is mapped.

About the MC IT (database) Role

There is also an IT database administrator (IT) role that controls a user's access to MC-managed databases. If you grant an MC user both IT roles, it means the user can perform some configuration on MC and also has access to one or more MC-managed databases. The database mapping is not required, but it gives the IT user wider privileges.

The two IT roles are similar, but they are not the same. The following table summarizes the primary difference between them, but see [IT Role \(db\)](#) for details.

MC configuration IT role	MC database IT role
Monitor MC-managed database, view non-database messages, and manage user access	Monitor databases on which the user has privileges, view the database overview and activity pages, monitor the node state view messages and mark them read/unread, view database settings. Can also be mapped to one or more Vertica databases.

NONE Role (mc)

The default role for all newly-created users on MC is NONE, which prevents users granted this role from configuring the MC. When you create MC users with the NONE role, you grant them an [MC database-level role](#). This assignment maps the MC user to a user account on a specific database and specifies that the NONE user inherits the database user's privileges to which he or she is mapped.

Which database-level role you grant this user with NONE privileges—whether ADMIN (db) or IT (db) or USER (db)—depends on the level of access you want the user to have on the MC-managed database. Database roles have no impact on the ADMIN and IT roles at the MC configuration level.

MC Database Privileges

When you [create MC users](#), you first assign them [MC configuration](#) privileges, which controls what they can do on the MC itself. In the same user-creation operation, you grant access to one or more MC-managed databases. MC database access does not give the MC user privileges directly on Vertica; it provides MC users varying levels of access to assigned database functionality through the MC interface.

Assign users an MC database level through one of the following roles:

- [ADMIN Role \(db\)](#)—Full access to all databases managed by MC. Actual privileges ADMINs inherit depend on the database user account used to create or import the Vertica database into the MC interface.
- [Associate Role \(Database\)](#)—Full access to all databases managed by MC. Cannot start, stop, or drop a database. Actual privileges that Associates receive depend on those defined for the database user account to which the Associate user is mapped.
- [IT Role \(db\)](#)—Can start and stop a database but cannot remove it from the MC interface or drop it.
- [USER Role \(db\)](#)—Can view database information through the database Overview and Activities pages but is restricted from viewing more detailed data.

Mapping MC Users to Database to Avoid Conflicts

When you assign an MC database level to an MC user, map the MC user account to a database user account to:

- Inherit the privileges assigned to that database user
- Prevent the MC user from doing or seeing anything not allowed by the privileges for the user account on the server database

Privileges assigned to the database user supersede privileges of the MC user if there is a conflict, such as stopping a database. When the MC user logs into MC using an MC user name and password, Vertica compares privileges for database-related activities to the privileges on the database account to which you mapped the MC user. Vertica allows operations in MC to the user only when that user has both MC privileges and corresponding database privileges.

Tip: As a best practice, you should identify, in advance, the appropriate Vertica database user account that has privileges or roles similar to one of the MC database roles.

See [Creating an MC User](#) and [Mapping an MC user to a database user's privileges](#) for more information.

MC Database Privileges By Role

The following table summarizes MC database-level privileges by user role. The table shows the default privileges each role has. Operations marked "database user privilege" are dependent on the privileges of the Vertica database user account to which the MC user is mapped.

Default database-level privileges	ADMIN	ASSOCIATE	IT	USER
View database Overview page	Yes	Yes	Yes	Yes
View database messages	Yes	Yes	Yes	Yes
Delete messages and mark read/unread	Yes	Yes	Yes	
Audit and install Vertica licenses	Database user privilege	Database user privilege		
View database Activity page: <ul style="list-style-type: none"> • Queries chart • Internal Sessions chart • User Sessions chart • System Bottlenecks chart • User Query Phases chart 	Yes	Database user privilege	Database user privilege	Database user privilege
View database Activity page: <ul style="list-style-type: none"> • Queries chart > Detail page • Table Treemap chart • Query Monitoring chart • Resource Pools Monitoring chart 	Database user privilege	Database user privilege		

Default database-level privileges	ADMIN	ASSOCIATE	IT	USER
Start a database	Yes			
Rebalance, stop, or drop databases	Database user privilege			
View Manage page	Yes	Yes	Yes	Yes
View node details	Yes	Yes	Yes	
Replace, add, or remove nodes	Database user privilege			
Start/stop a node	Yes			
View database Settings page	Yes	Yes	Yes	
Modify database Settings page	Database user privilege	Database user privilege		
View Database Designer	Database user privilege	Database user privilege		

ADMIN Role (db)

ADMIN is a superuser with full privileges to monitor MC-managed database activity and messages. Other database privileges (such as stop or drop the database) are governed by the user account on the Vertica database that this ADMIN (db) user is mapped to. ADMIN is the most permissive role and is a superset of privileges granted to the [Associate Role \(Database\)](#), [IT](#), and [USER](#) roles.

The ADMIN user has the following database privileges by default:

- View the database Overview page
- View and delete database messages
- Mark messages read or unread
- Start the database

- View the Manage page
- View node details
- Start or stop a node
- View database settings
- View the following database Activity page charts:
 - Queries
 - Internal Sessions
 - User Sessions
 - System Bottlenecks
 - User Query Phases

The following database operations depend on the database user's role that you mapped this ADMIN user to:

- Install or audit a license
- Rebalance, stop, or drop databases
- Add, replace, or remove nodes
- Manage database settings
- View Database Designer
- View additional information on the database Activity page:
 - Detailed information in the Queries chart Detail page
 - Table Treemap chart
 - Query Monitoring chart
 - Resource Pools Monitoring chart

Note: Database access granted through Management Console never overrides roles granted on a specific Vertica database.

About the ADMIN (MC configuration) Role

There is also an MC configuration administrator role that defines what the user can change on the MC itself. The two ADMIN roles are not the same. Unlike the MC configuration role of ADMIN, which can manage all MC users and all databases imported into the UI, the MC database ADMIN role has privileges only on the databases you map this user to. The following table summarizes the primary difference between them. See [ADMIN Role \(mc\)](#) for additional details.

MC database ADMIN role	MC configuration ADMIN role
Perform database-specific activities, such as stop and start the database, and monitor query and user activity and resources. Other database operations depend on that user's privileges on the specific database. This ADMIN role cannot configure MC.	Perform all administrative operations on the MC itself, including restarting the MC process. Privileges extend to monitoring all databases created and imported by MC. Anything database-related beyond that scope depends on the user's privileges granted on the database through GRANT statements.

Associate Role (Database)

The Associate role is an MC database access role. It is similar to the Admin role. It has privileges to monitor activity and messages on databases managed by MC. Unlike Admin users, Associate users cannot start, stop, or drop the database. The Associate user role is mapped to a user account on the database. This mapped user role determines what other database privileges the Associate role has (such as modifying settings, installing licenses, and viewing the database designer).

The Associate user has the following database privileges by default:

- View the database Overview page
- View and delete database messages
- Mark messages read or unread
- View the Manage page
- View node details
- View database settings

The following database operations depend on the database user's role that you mapped this Associate user to:

- Install or audit a license
- Manage database settings
- View Database Designer
- View the database Activity page

Note: Database access granted through Management Console never overrides roles granted on a specific Vertica database.

IT Role (db)

IT can view most details about an MC-managed database, such as messages (and mark them read/unread), the database overall health and activity/resources, cluster and node state, and MC settings. You grant and manage user role assignments through the **MC Settings > User management** page on the MC.

About the IT (MC configuration) Role

There is also an IT role at the MC configuration access level. The two IT roles are similar, but they are not the same. If you grant an MC user both IT roles, it means the user can perform some configuration on MC and also has access to one or more MC-managed databases. The following table summarizes the primary difference between them, but see [IT Role \(mc\)](#) for additional details.

MC database IT	MC configuration IT
Monitor databases on which the user has privileges, view the database overview and activity pages, monitor the node state view messages and mark them read/unread, view database settings.	Monitor MC-managed database, view non-database messages, and manage user access.

USER Role (db)

USER has limited database privileges, such as viewing database cluster health, activity/resources, and messages. MC users granted the USER database role might have higher

levels of permission on the MC itself, such as the [IT Role \(mc\)](#). Alternatively, USER users might have no (NONE) privileges to configure MC. How you combine the two levels is up to you.

Granting Database Access to MC Users

If you did not grant an MC user a [database-level role](#) when you created the user account, you can do so in the User Management tab in MC Settings.

Granting the user an MC database-level role associates the MC user with a database user's privileges and ensures that the MC user cannot do or see anything not allowed by the privileges set up for the user account on the server database. When that MC user logs in to MC, his or her MC privileges for database-related activities are compared to that user's privileges on the database itself. Only when the user has both MC privileges and corresponding database privileges will the operations be exposed in the MC interface.

Prerequisites

Before you grant database access to an MC user, see the prerequisites in [Creating an MC User](#).

Grant a Database-Level Role to an MC user

1. Log in to Management Console as an administrator and navigate to **MC Settings > User management**.
2. Select an MC user and click **Edit**.
3. Verify the [MC Configuration Privileges](#) are what you want them to be. NONE is the default.
4. Next to the **DB access levels section**, click **Add** and provide the following database access credentials:
 - i. **Choose a database.** Select a database from the list of MC-discovered (databases that were created on or imported into the MC interface).
 - ii. **Database username.** Enter an existing database user name or, if the database is running, click the ellipses [...] to browse for a list of database users, and select a name from the list.
 - iii. **Database password.** Enter the password to the database user account (not this username's password).

- iv. **Restricted access.** Chose a database level ([ADMIN](#), [IT](#), or [USER](#)) for this user.
 - v. Click **OK** to close the **Add permissions** dialog box.
5. Optionally change the user's **Status** (enabled is the default).
6. Click **Save**.

How MC Validates New Users

After you click OK to close the Add permissions dialog box, MC tries to validate the database username and password entered against the selected MC-managed database or against your organization's LDAP directory. If the credentials are found to be invalid, you are asked to re-enter them.

If the database is not available at the time you create the new user, MC saves the username/password and prompts for validation when the user accesses the Database and Clusters page later.

Managing Client Connections

Each client session to MC uses a connection from `MaxClientSessions`, a database configuration parameter. This parameter determines the maximum number of sessions that can run on a single database cluster node. Sometimes multiple MC users, mapped to the same database account, are concurrently monitoring the Overview and Activity pages. In such cases, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for `MaxClientSessions` on an MC-monitored database to account for extra sessions. See [Managing Sessions](#) for details.

Managing Database Clusters

To perform database/cluster-specific tasks on one or more MC-managed clusters, navigate to the **Databases and Clusters** page.

MC administrators see the Import/Create Database Cluster options, while non-administrative MC users see only the databases on which they have been assigned the appropriate [access levels](#). Depending on your access level, the database-related operations you can perform on the MC interface include:

- [Create a new database/cluster](#).
- [Import an existing database/cluster](#) into the MC interface.
- Start the database, unless it is already running (green).
- Stop the database, but only if no users are connected.
- Remove the database from the MC interface.

Note: Remove does not drop the database. A Remove operation leaves it in the cluster, hidden from the UI. To add the database back to the MC interface, import it using the IP address of any cluster node. A Remove operation also stops metrics gathering on that database, but statistics gathering automatically resumes after you re-import.

- Drop the database after you ensure no users are connected. Drop is a permanent action that drops the database from the cluster.
- View the Overview page, a dashboard view into the health of your database cluster. From this page you can drill down into more detailed database-specific information by clicking data points in the graphs.
- View the Manage page, which shows all nodes in the cluster, as well as each node's state. You can also see a list of monitored databases on the selected cluster and its state. For example, a green arrow indicates a database in an UP state. For node-specific information, click any node to open the Node Details page.

Create an Empty Database Using MC

You can create a new database on an existing Vertica cluster through the Management Console interface.

Database creation can be a long-running process, lasting from minutes to hours, depending on the size of the target database. You can close the web browser during the process and sign back in to MC later; the creation process continues unless an unexpected error occurs. See the **Notes** section below the procedure on this page.

You currently need to use command line scripts to define the database schema and load data. Refer to the topics in [Configuration Procedure](#). You should also run the Database Designer, which you access through the Administration Tools, to create either a comprehensive or incremental design. Consider using the [Tutorial](#) in Getting Started to create a sample database you can start monitoring immediately.

How to Create an Empty Database on an MC-managed Cluster

1. If you are already on the **Databases and Clusters** page, skip to the next step; otherwise:
 - a. [Connect](#) to MC and sign in as an MC administrator.
 - b. On the [Home page](#), click **Existing Infrastructure** to view the Databases and Clusters page.
2. If no databases exist on the cluster, continue to the next step; otherwise:
 - a. If a database is running on the cluster on which you want to add a new database, select the database and click **Stop**.
 - b. Wait for the running database to have a status of *Stopped*.
3. Click the cluster on which you want to create the new database and click **Create Database**.
4. The Create Database wizard opens. Provide the following information:

- Database name and password. See [Creating a Database Name and Password](#) for rules.
- Optionally click **Advanced** to open the advanced settings and change the port, catalog path, and data path. By default the MC application/web server port is 5450 and paths are /home/dbadmin, or whatever you defined for the paths when you ran the Cluster Creation Wizard or the `install_vertica` script. Do not use the default agent port 5444 as a new setting for the MC port. See **MC Settings > Configuration** for port values.

5. Click **Continue**.

6. Select nodes to include in the database.

The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node and clear **Include host in new database** to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the **Include** check box.

7. Click **Create** in the **Database Configuration** window to create the database on the nodes.

The creation process takes a few moments, after which the database starts and a **Success** message appears on the interface.

8. Click **OK** to close the success message.

The Manage page opens and displays the database nodes. Nodes not included in the database are colored gray, which means they are standby nodes you can include later. To add nodes to or remove nodes from your Vertica cluster, which are not shown in standby mode, you must run the `install_vertica` script.

Notes

- If warnings occur during database creation, nodes will be marked on the UI with an Alert icon and a message.
 - Warnings do not prevent the database from being created, but you should address warnings after the database creation process completes by viewing the database **Message Center** from the MC Home page.

- Failure messages display on the database **Manage** page with a link to more detailed information and a hint with an actionable task that you must complete before you can continue. Problem nodes are colored red for quick identification.
- To view more detailed information about a node in the cluster, double-click the node from the Manage page, which opens the **Node Details** page.
- To create MC users and grant them access to an MC-managed database, see [About MC Users](#) and [Creating an MC User](#).

See Also

- [Creating a Cluster Using MC](#)
- [Troubleshooting with MC Diagnostics](#)
- [Restarting MC](#)

Import an Existing Database Into MC

If you have already upgraded your database to the current version of Vertica, MC automatically discovers the cluster and any databases installed on it, regardless of whether those databases are currently running or are down.

Note: If you haven't created a database and want to create one through the MC, see [Create an Empty Database Using MC](#).

How to Import an Existing Database on the Cluster

The following procedure describes how to import an existing database into the MC interface so you can monitor it.

1. [Connect](#) to Management Console and sign in as an MC administrator.
2. On the MC [Home page](#), click **Existing Infrastructure**.

3. On the Databases and Clusters page, click the cluster and click **View** in the dialog box that opens.
4. On the left side of the page under the Databases heading, and click **Import Discovered**.

Tip: A running database appears as Monitored, and any non-running databases appear as Discovered. MC supports only one running database on a single cluster at a time. You must shut down a running database on a cluster in order to monitor another database on that cluster.

5. In the **Import Database** dialog box:
 - a. Select the database you want to import.
 - b. Optionally clear auto-discovered databases you don't want to import.
 - c. Supply the database administrator username and password and click **Import**.
(Supplying a non-administrator username prevents MC from displaying some charts after import.)

After Management Console connects to the database it opens the **Manage** page, which provides a view of the cluster nodes. See [Monitoring Cluster Status](#) for more information.

You perform the import process once per existing database. Next time you connect to Management Console, your database appears under the Recent Databases section on the Home page, as well as on the Databases and Clusters page.

Note: The system clocks in your cluster must be synchronized with the system that is running Management Console to allow automatic discovery of local clusters.

Import and Monitor in a Hadoop Environment

You can use Management Console to connect to and monitor a Vertica database that resides in an Apache Hadoop environment.

Connecting to Ambari and Importing Vertica Within a Hadoop Environment

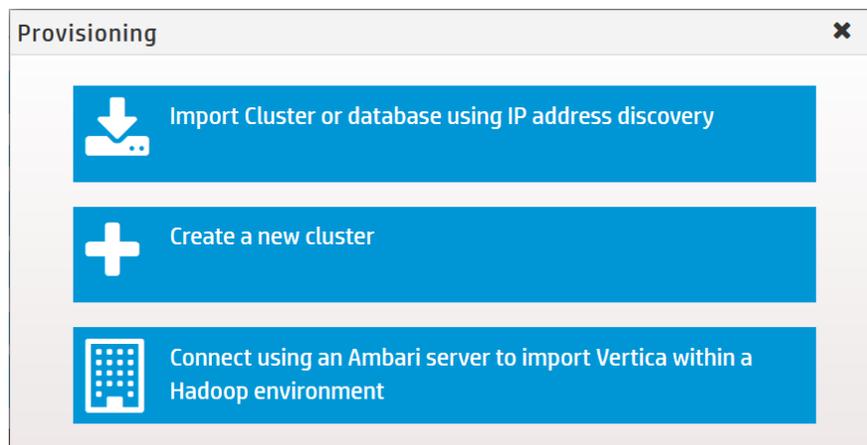
To import your Vertica database that resides in a Hadoop environment, connect to that Hadoop environment in Management Console through an Apache Ambari server. Then, through that connection, import Vertica.

Before you connect and import, you must:

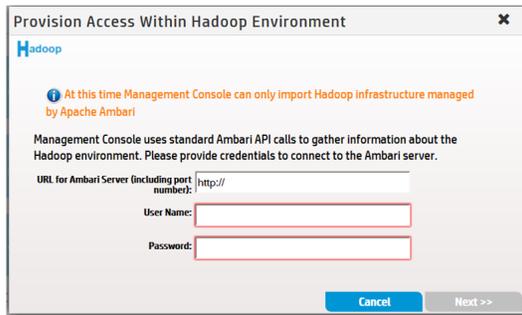
- Install Vertica on a Hadoop cluster
- Install Apache Ambari version 1.6.1 or 2.1.0
- Enable Ganglia on your Hadoop cluster, to get the most information from your Hadoop environment

To connect to an Ambari server:

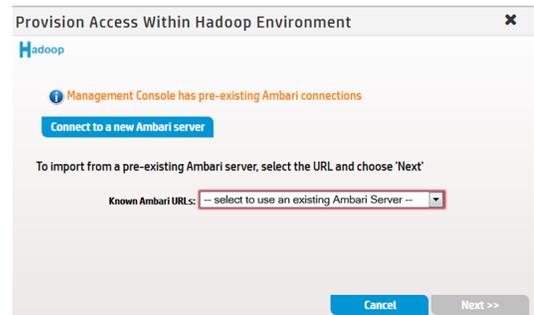
1. From the Management Console home page, select **Provisioning**. Then, click **Connect using an Ambari server to import Vertica within a Hadoop environment**.



2. In the **Provision Access within Hadoop Environment** window, enter a new Ambari connection with your username and password . If have already have an existing connection,a window opens that allows you to select an existing connection from the drop-down menu.



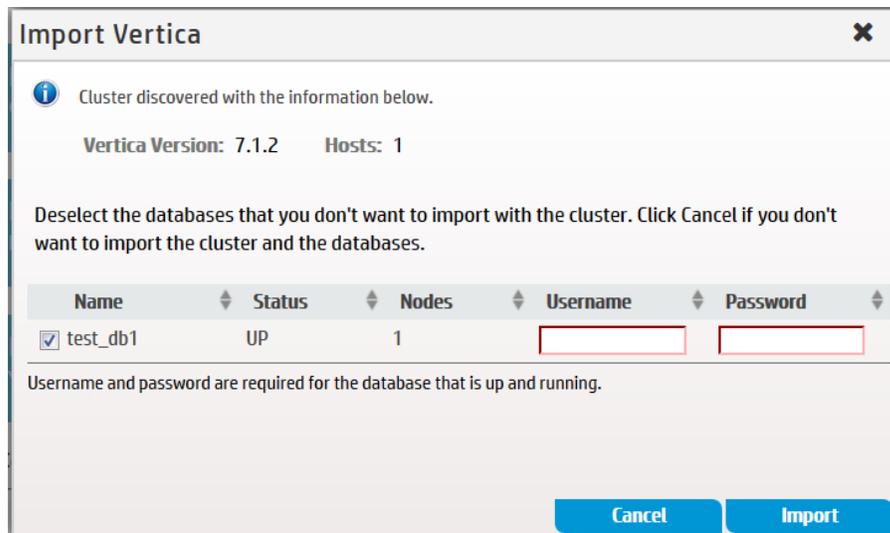
OR



3. In the next window, select the Hadoop cluster where the Vertica database you want to monitor resides. MC will discover Hadoop clusters that are currently monitored by the Ambari server you specify.

A window appears confirming that your Hadoop cluster is saved. If MC currently does not monitor Vertica clusters in the specified Hadoop environment, you have the option to import Vertica clusters at this time.

4. Enter the IP address for the Vertica database you want to import and monitor. If Vertica is running on multiple hosts, enter the IP address of one of them.
5. Enter the API key for the Vertica cluster. The API key is generated during Vertica installation and you can find it in the `/opt/vertica/config/apikeys.dat` file.
6. When the next window displays the discovered databases, select one or more database you want to import, along with the database username and password.

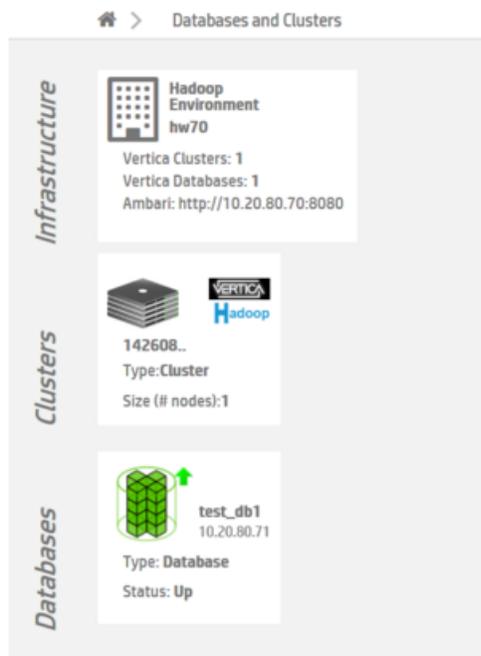


7. Confirm that the import is successful. If it is, a success message appears. Click **Done** to go to the **Existing Infrastructure** page.

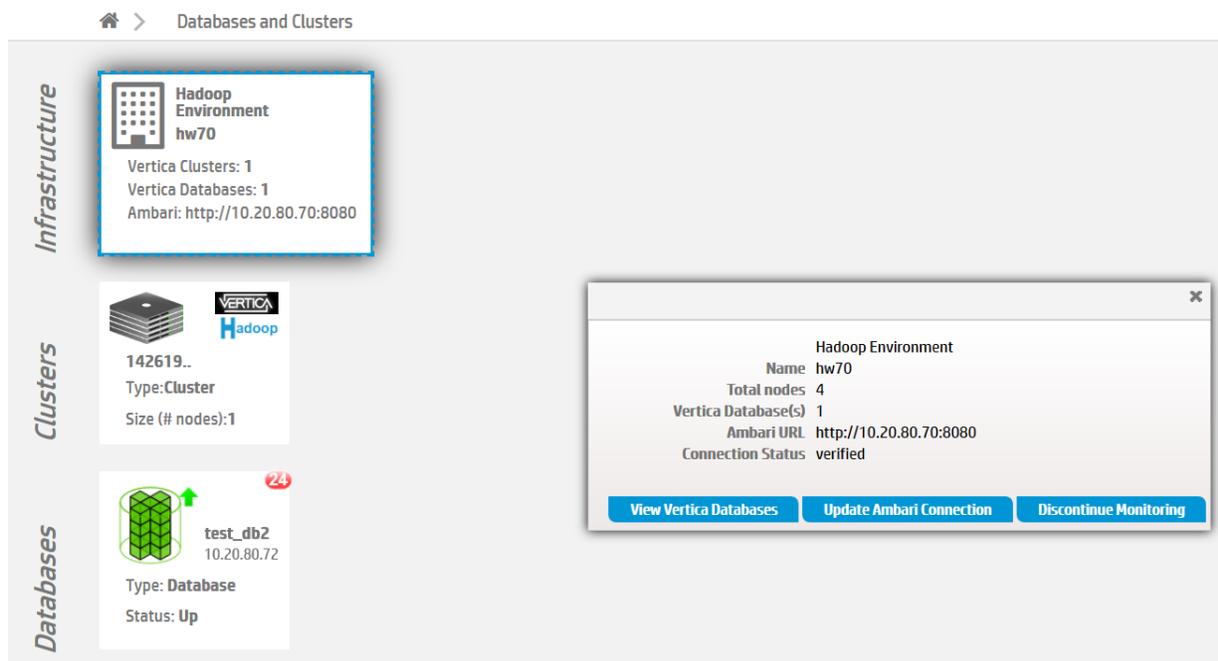
To import an additional Vertica cluster within a Hadoop environment, click **Import Cluster or database using IP address discovery** under **Provisioning**. Management Console will automatically associate the cluster with the existing Hadoop environment.

Monitoring Vertica Within a Hadoop Environment

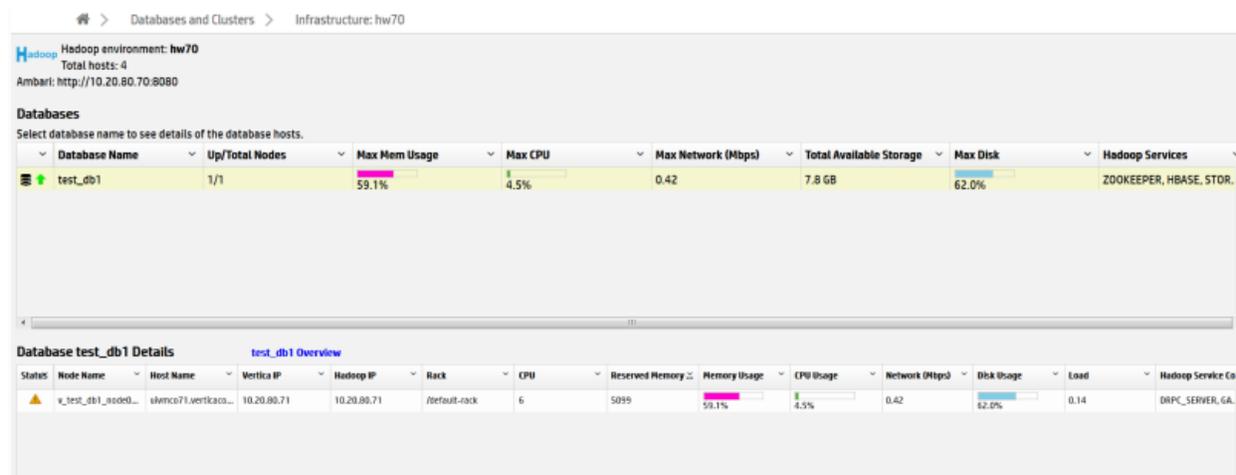
To monitor the Vertica clusters in a Hadoop environment, navigate to the **Existing Infrastructure** page:



Click to select the Hadoop environment, and then click **View Vertica Databases**.



The resulting screen displays information about the Vertica databases that reside in a Hadoop environment:



You can monitor information like resource usage, Hadoop services, and database and connection status.

Updating and Removing an Ambari Connection

To update or remove an existing Ambari connection, go to the MC **Existing Infrastructure** page, and click on the relevant Hadoop environment.

To update a connection, click **Update Ambari Connection**. Step through the wizard to update the connection.

To remove a connection, select **Update Ambari Connection** and choose **Remove Connection**, or click **Discontinue Monitoring** and then confirm that you want to remove the connection. Removing the connection also removes all Vertica databases associated with this connection from MC monitoring. You can re-import the databases later if needed.

See Also:

[Integrating with Apache Hadoop](#)

Managing MC Settings

The **MC Settings** page allows you to configure properties specific to Management Console. You can:

- Change MC and agent default port assignments.
- Enable and disable username and password auto-complete at MC login. (After disabling, clear your browser's cache.)
- Control the following monitoring settings in MC:
 - Enable checks and set alert thresholds for spread retransmit rate. This setting is disabled by default. The recommended alert threshold for spread retransmit rate is 10%.
 - Set alert thresholds for free MC disk space checks. The recommended alert threshold is 500 MB.
 - Exclude MC queries from activity charts.
 - Set refresh intervals for MC charts and pages.
- Upload a new SSL certificate.
- Use LDAP for user authentication.
- Create new MC users and, with their user credentials, map them to an database managed by MC on the Vertica server. See [Creating an MC User](#) and [Managing MC Users](#).
- View your version of Vertica or upload a new Vertica binary file.
- Customize the look and feel of MC with themes. See [Customizing Look and Feel](#).
- Configure MC to use an alternative data source to monitor your database. See [Monitoring External Data Sources in Management Console](#).
- Enable MC to send email alerts. See [Set Up Email](#) .
- Configure [Extended Monitoring](#), which allows you to monitor more long-term data in MC:
 - Set up an external storage database for Extended Monitoring. See [Managing the Storage Database](#).
 - Enable or disable Extended Monitoring on your databases. See [Managing Extended Monitoring on a Database](#).

Modifying Database-Specific Settings

To inspect or modify settings related to a database managed by MC, go to the **Existing Infrastructure** page. On this page, select a running database to see its Overview page. From the bottom of the Overview page, click the Settings tab to make modifications to database-specific settings.

Customizing Look and Feel

Management Console themes provide a unique look and feel to the Management Console interface. Access these themes through the Theme page in MC Settings.

Changing Themes

Only administrators, who have access to MC Settings, can alter themes. The selected theme is visible to all Management Console users.

To apply a new theme on Management Console, go to MC Settings > Theme. On the Theme page, select a theme from the drop-down menu. A preview of the selected theme appears. Click Apply in the upper right hand of the page to apply the change.

Vertica Management Console dbadmin Log out  32 

[Home](#) > Management Console settings: Theme **Apply** **Done**

Configuration	Select a theme <input type="text" value="HPE Theme"/> Don't forget to click 'Apply'!
Monitoring	
SSL Certificates	
Authentication	
User Management	
Vertica Installation	
Theme	
Data Source	
Email Gateway	
MC Storage DB Setup	
Extended Monitoring	

Available Themes

The Micro Focus Theme is the default Management Console theme. This theme includes the colors and styles used by Micro Focus branding.

Prepare data and databases



Provisioning
Create and import databases and clusters

Manage Information



Infrastructure
Manage databases and clusters



Message Center
Go to the database message center



Diagnostics
View diagnostics and support information



MC Settings
Manage application and user settings

Recent Databases

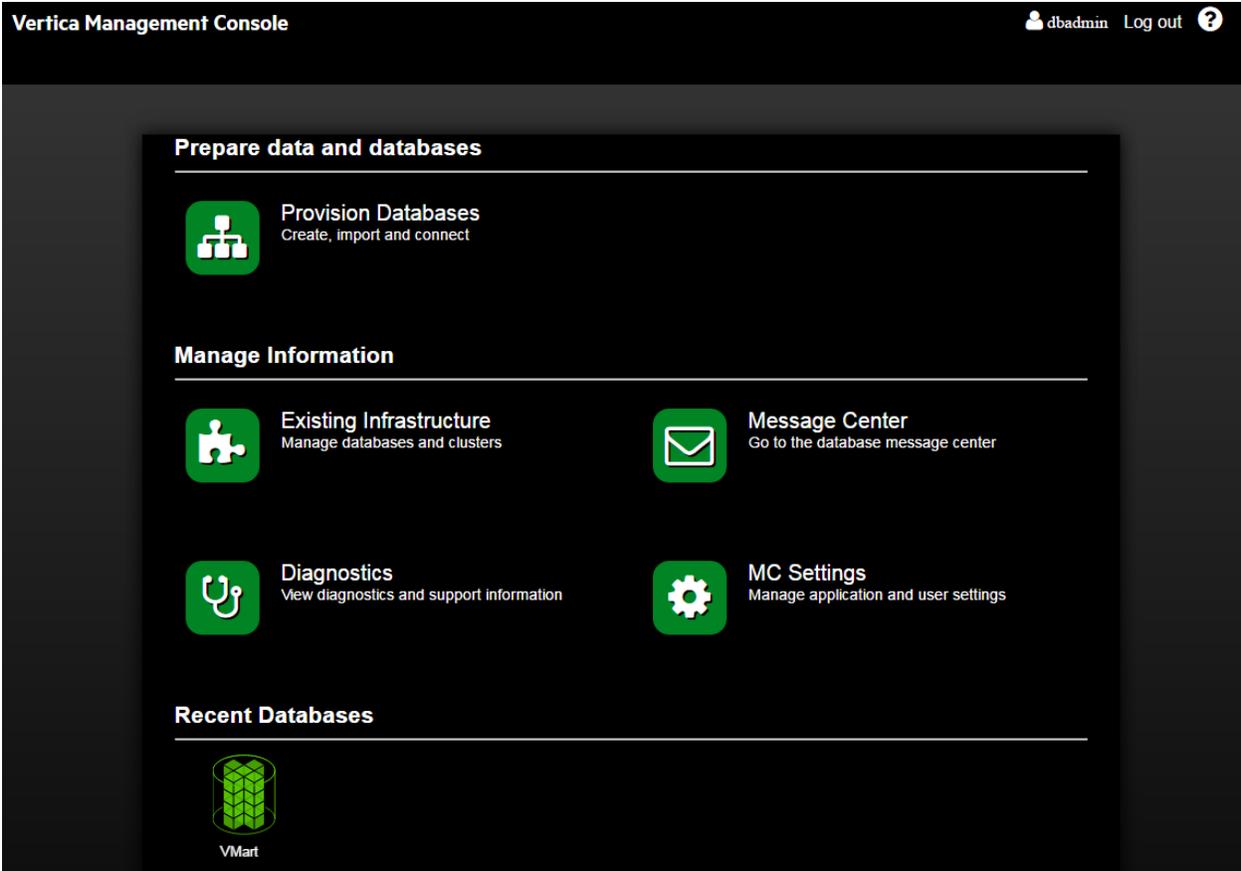


doccdb



VMart_BR

The Vertica Classic theme features a dark background and green accents, evoking classic computer terminal interfaces.



Changing MC or Agent Ports

When you configure MC, the Configuration Wizard sets up the following default ports:

- 5450—Used to connect a web browser session to MC and allows communication from Vertica cluster nodes to the MC application/web server
- 5444—Provides MC-to-node and node-to-node (agent) communications for database create/import and monitoring activities

If You Need to Change the MC Default Ports

A scenario might arise where you need to change the default port assignments for MC or its agents. For example, perhaps one of the default ports is not available on your Vertica cluster, or you encounter connection problems between MC and the agents. The following topics describe how to change port assignments for MC or its agents.

See Also

- [Ensure Ports Are Available](#)

How to Change the Agent Port

Changing the agent port takes place in two steps: at the command line, where you modify the `config.py` file and through a browser, where you modify MC settings.

Change the Agent Port in `config.py`

1. Log in as root on any cluster node and change to the agent directory:

```
# cd /opt/vertica/agent
```

2. Use any text editor to open `config.py`.

3. Scroll down to the `agent_port = 5444` entry and replace 5444 with a different port number.
4. Save and close the file.
5. Copy `config.py` to the `/opt/vertica/agent` directory on all nodes in the cluster.
6. Restart the agent process by running the following command:

```
# /etc/init.d/vertica_agent restart
```

Note: If you are using Red Hat Enterprise Linux/CentOS 7, use the following command instead:

```
# /opt/vertica/sbin/vertica_agent restart
```

7. Repeat (as root) Step 6 on each cluster node where you copied the `config.py` file.

Change the Agent Port on MC

1. Open a web browser and [connect to MC](#) as a user with [MC ADMIN](#) privileges.
2. Navigate to **MC Settings > Configuration**.
3. Change Default Vertica agent port from 5444 to the new value you specified in the `config.py` file.
4. Click **Apply** and click **Done**.
5. Restart MC so MC can connect to the agent at its new port. See [Restarting MC](#).

How to Change the MC Port

Use this procedure to change the default port for MC's application server from 5450 to a different value.

1. Open a web browser and [connect to MC](#) as a user with [MC ADMIN](#) privileges.
2. On the MC Home page, navigate to **MC Settings > Configuration** and change the *Application server running port* value from 5450 to a new value.
3. In the change-port dialog, click **OK**.
4. [Restart MC](#).
5. Reconnect your browser session using the new port. For example, if you changed the port from 5450 to 5555, use one of the following formats:

```
https://00.00.00.00:5555/
```

OR

```
https://hostname:5555/
```

Backing Up MC

Before you [upgrade MC](#), Micro Focus recommends that you back up your MC metadata (configuration and user settings). Use a storage location external to the server on which you installed MC.

1. On the target server (where you want to store MC metadata), log in as root or a user with sudo privileges.
2. Create a backup directory as in following example:

```
# mkdir /backups/mc/mc-backup-20130425
```

3. Copy the /opt/vconsole directory to the new backup folder:

```
# cp -r /opt/vconsole /backups/mc/mc-backup-20130425
```

Troubleshooting with MC Diagnostics

The Management Console **Diagnostics** page, which you access from the Home page, helps you resolve issues within the MC process, not the database.

What You Can Diagnose

- View Management Console logs, which you can sort by column headings, such as type, component, or message).
- [Search](#) within messages for key words or phrases and search for log entries within a specific time frame.
- [Export](#) database messages to a file.
- Reset console parameters to their original configuration.

Caution: Reset removes all data (monitoring and configuration information) from storage and forces you to [reconfigure MC](#) as if it were the first time.

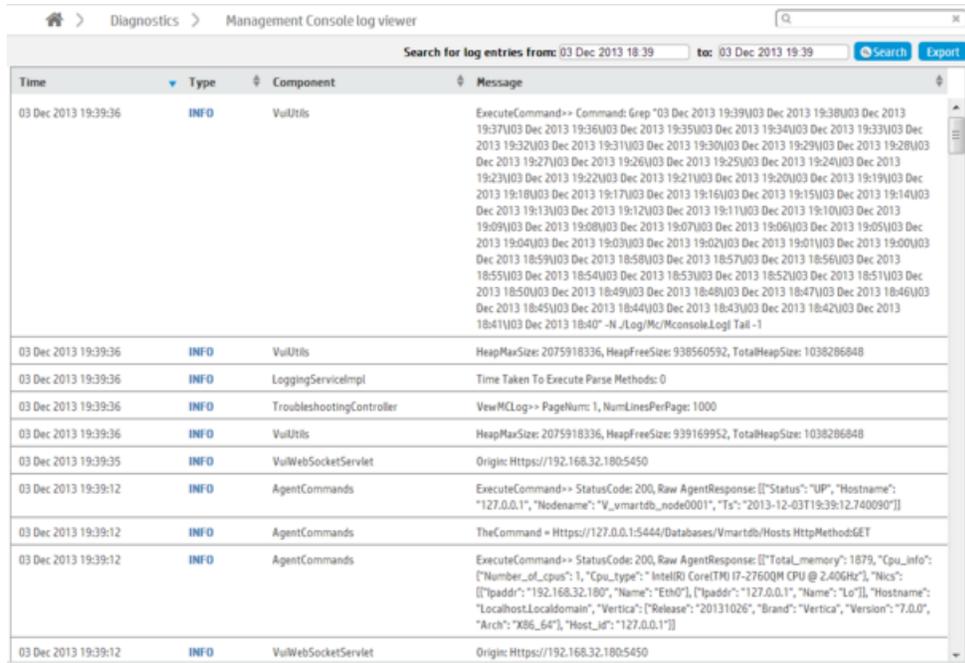
- [Restart the Management Console process](#). When the process completes, you are directed back to the login page.

Viewing the MC Log

If you want to browse MC logs (not database logs), navigate to the **Diagnostics > MC Log** page.

This page provides a tabular view of the contents at `/opt/vconsole/log/mc/mconsole.log`, letting you more easily identify and troubleshoot issues related to MC.

You can sort log entries by clicking the column header and search within messages for key words, phrases, and log entries within a specific time frame. You can also export log messages to a file.



The screenshot shows the Management Console log viewer interface. At the top, there is a search bar with the text "Search for log entries from: 03 Dec 2013 18:39 to: 03 Dec 2013 19:39" and buttons for "Search" and "Export". Below the search bar is a table with the following columns: Time, Type, Component, and Message. The table contains several log entries, including:

Time	Type	Component	Message
03 Dec 2013 19:39:36	INFO	VulUtils	ExecuteCommand-> Command: Grep *03 Dec 2013 19:39:03 Dec 2013 19:38:03 Dec 2013 19:37:03 Dec 2013 19:36:03 Dec 2013 19:35:03 Dec 2013 19:34:03 Dec 2013 19:33:03 Dec 2013 19:32:03 Dec 2013 19:31:03 Dec 2013 19:30:03 Dec 2013 19:29:03 Dec 2013 19:28:03 Dec 2013 19:27:03 Dec 2013 19:26:03 Dec 2013 19:25:03 Dec 2013 19:24:03 Dec 2013 19:23:03 Dec 2013 19:22:03 Dec 2013 19:21:03 Dec 2013 19:20:03 Dec 2013 19:19:03 Dec 2013 19:18:03 Dec 2013 19:17:03 Dec 2013 19:16:03 Dec 2013 19:15:03 Dec 2013 19:14:03 Dec 2013 19:13:03 Dec 2013 19:12:03 Dec 2013 19:11:03 Dec 2013 19:10:03 Dec 2013 19:09:03 Dec 2013 19:08:03 Dec 2013 19:07:03 Dec 2013 19:06:03 Dec 2013 19:05:03 Dec 2013 19:04:03 Dec 2013 19:03:03 Dec 2013 19:02:03 Dec 2013 19:01:03 Dec 2013 19:00:03 Dec 2013 18:59:03 Dec 2013 18:58:03 Dec 2013 18:57:03 Dec 2013 18:56:03 Dec 2013 18:55:03 Dec 2013 18:54:03 Dec 2013 18:53:03 Dec 2013 18:52:03 Dec 2013 18:51:03 Dec 2013 18:50:03 Dec 2013 18:49:03 Dec 2013 18:48:03 Dec 2013 18:47:03 Dec 2013 18:46:03 Dec 2013 18:45:03 Dec 2013 18:44:03 Dec 2013 18:43:03 Dec 2013 18:42:03 Dec 2013 18:41:03 Dec 2013 18:40 -N./Log/Mc/Mconsole.Log Tail -1
03 Dec 2013 19:39:36	INFO	VulUtils	HeapMaxSize: 2075918336, HeapFreeSize: 938560592, TotalHeapSize: 1038286848
03 Dec 2013 19:39:36	INFO	LoggingServiceImpl	Time Taken To Execute Parse Methods: 0
03 Dec 2013 19:39:36	INFO	TroubleshootingController	ViewMCLog-> PageNum: 1, NumLinesPerPage: 1000
03 Dec 2013 19:39:36	INFO	VulUtils	HeapMaxSize: 2075918336, HeapFreeSize: 939169952, TotalHeapSize: 1038286848
03 Dec 2013 19:39:35	INFO	VulWebSocketServlet	Origin: https://192.168.32.180:5450
03 Dec 2013 19:39:12	INFO	AgentCommands	ExecuteCommand-> StatusCode: 200, Raw AgentResponse: [{"Status": "UP", "Hostname": "127.0.0.1", "Nodename": "V_vmartdb_node0001", "Ts": "2013-12-03T19:39:12.740090"}]
03 Dec 2013 19:39:12	INFO	AgentCommands	TheCommand = https://127.0.0.1:5444/Databases/Vmartdb/Hosts HttpMethod:GET
03 Dec 2013 19:39:12	INFO	AgentCommands	ExecuteCommand-> StatusCode: 200, Raw AgentResponse: [{"Total_memory": 1879, "Cpu_info": {"Number_of_cpus": 1, "Cpu_type": "Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz"}, "Nics": [{"Ipaddr": "192.168.32.180", "Name": "Eth0"}, {"Ipaddr": "127.0.0.1", "Name": "Lo"}], "Hostname": "Localhost.Localdomain", "Vertica": {"Release": "20131026", "Brand": "Vertica", "Version": "7.0.0", "Arch": "X86_64", "Host_id": "127.0.0.1"}]
03 Dec 2013 19:39:12	INFO	VulWebSocketServlet	Origin: https://192.168.32.180:5450

Exporting the User Audit Log

When an MC user makes changes on Management Console, whether to an MC-managed database or to the MC itself, their action generates a log entry that contains data you can export to a file.

If you perform an MC factory reset (restore MC to its pre-configured state), you automatically have the opportunity to export audit records before the reset occurs.

To Manually Export MC User Activity

1. From the MC Home page, click **Diagnostics** and then click **Audit Log**.
2. On the Audit log viewer page, click **Export** and save the file to a location on the server.

To see what types of user operations the audit logger records, see [Monitoring MC User Activity Using Audit Log](#).

Restarting MC

You might need to restart the MC web/application server for a number of reasons, such as after you change port assignments, use the MC interface to import a new SSL certificate, or if the MC interface or Vertica-related tasks become unresponsive.

Restarting MC requires [ADMIN Role \(mc\)](#) or [SUPER Role \(mc\)](#) privileges.

How to Restart MC through the MC Interface (Using Your Browser)

1. Open a web browser and [connect to MC](#) as an administrator.
2. On MC's Home page, click **Diagnostics**.
3. Click **Restart Console** and then click OK to continue or Cancel to return to the Diagnostics page..

The MC process shuts down for a few seconds and automatically restarts. After the process completes, you are directed back to the sign-in page.

How to Restart MC at the Command Line

If you are unable to connect to MC through a web browser for any reason, such as if the MC interface or Vertica-related tasks become unresponsive, you can run the `vertica-consoled` script with `start`, `stop`, or `restart` arguments.

Follow these steps to start, stop, or restart MC.

1. As root, open a terminal window on the server on which MC is installed.
2. Run the `vertica-consoled` script:

```
# /etc/init.d/vertica-consoled { stop | start | restart }
```

For versions CentOS 7 and above, run:

```
# systemctl { stop | start | restart } vertica-consoled
```

Important: The `systemctl` function requires you to both start and stop services explicitly. If you kill or stop the `vertica-consoled` process without using `systemctl stop`, you cannot start the MC process again with the original `systemctl start` command. Instead, you must run `systemctl stop vertica-consoled` before running `systemctl start vertica-consoled`.

<code>stop</code>	Stops the MC application/web server.
<code>start</code>	Starts the MC application/web server. Caution: Use <code>start</code> only if you are certain MC is not already running. As a best practice, stop MC before you issue the <code>start</code> command.
<code>restart</code>	Restarts the MC application/web server. This process will report that the stop didn't work if MC is not already running.

How to Restart MC on an AMI

You can use the following steps to restart an MC AMI instance.

1. SSH into the MC host as user `dbadmin`:

```
$ ssh -i example.pem dbadmin@52.xx.xx.xx
```

2. Run the `vertica-consoled` script using `sudo`:

```
# sudo /etc/init.d/vertica-consoled { stop | start | restart }
```

Starting over

If you need to return MC to its original state (a "factory reset"), see [Resetting MC to Pre-Configured State](#).

Resetting MC to Pre-Configured State

If you decide to reset MC to its original, preconfigured state, you can do so on the **Diagnostics** page by clicking **Factory Reset**.

Tip: Consider trying one of the options described in [Restarting MC](#) first.

A factory reset removes all metadata (about a week's worth of database monitoring/configuration information and MC users) from storage and forces you to reconfigure MC again, as described in [Configuring MC](#) in *Installing Vertica*.

After you click **Factory Reset**, you have the chance to export audit records to a file by clicking **Yes**. If you click **No** (do not export audit records), the process begins. There is no undo.

Keep the following in mind concerning user accounts and the MC.

- When you first configure MC, during the configuration process you create an MC superuser (a Linux account). Issuing a **Factory Reset** on the MC does not create a new MC superuser, nor does it delete the existing MC superuser. When initializing after a **Factory Reset**, you must logon using the original MC superuser account.
- Note that, once MC is configured, you can add users that are specific to MC. Users created through the MC interface are MC specific. When you subsequently change a password through the MC, you only change the password for the specific MC user. Passwords external to MC (i.e., system Linux users and Vertica database passwords) remain unchanged.

For information on MC users, refer to the sections, [Creating an MC User](#) and [MC configuration privileges](#).

Avoiding MC Self-Signed Certificate Expiration

When you [connect to MC](#) through a client browser, Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your Vertica cluster, and on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See [Set Up Time Synchronization](#) in *Installing Vertica*.

Running Queries in Management Console

You can use the Query Runner to run SQL queries on your database through Management Console (MC). After executing a query, you can also get the query plan and profile information for the query on this page.

To reach the Query Runner, select your database from the Home page or the Databases and Clusters page to view your database's Overview page. Select Query Execution at the bottom of the Overview page.

Query History

Filter previous queries

```
1 SELECT sales_quantity, sales_dollar_amount, transaction_type, cc_name
2 FROM online_sales.online_sales_fact
3 INNER JOIN online_sales.call_center_dimension
4 ON (online_sales.online_sales_fact.call_center_key
5     = online_sales.call_center_dimension.call_center_key
6     AND sale_date_key = 156)
7 ORDER BY sales_dollar_amount DESC;
8 SELECT order_number, date_ordered
9 FROM store.store_orders_fact orders
10 WHERE orders.store_key IN (
11     SELECT store_key
12     FROM store.store_dimension
13     WHERE store_state = 'MA')
14     AND orders.vendor_key NOT IN (
15     SELECT vendor_key
16     FROM public.vendor_dimension
17     WHERE vendor_state = 'MA')
18     AND date_ordered < '2012-03-01';
19 SELECT store_key, order_number, date_ordered
```

Execute Queries

SELECT sales_qu SELECT order_nu SELECT store_ke

Query Results Query Plan Query Profile Export Data Auto-Resize all columns Search query results

sales_quantity	sales_dollar_amount	transaction_type	cc_name
7	589	purchase	Central Midwest
8	589	purchase	South Midwest
8	589	purchase	California

2514 rows | Execution time: 0.113s

Overview Activity Manage Design Load **Query Execution** Query Plan License Settings

+

To familiarize yourself with how queries work in Vertica, you can refer to the [Queries](#) section of the documentation, as well as the [SQL Reference Manual](#).

Limitations

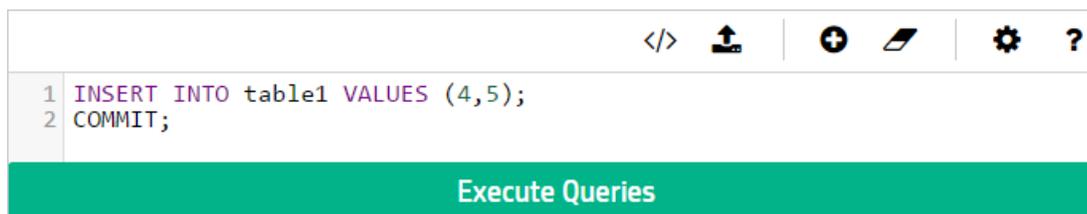
You cannot execute COPY LOCAL statements using the Query Runner. To do so, use the vsq client installed on the server. See [Using vsq](#). (To use MC to import data from Amazon S3

storage to your Vertica database, see [Loading Data Using MC.](#))

Manually commit any transactions (INSERT and COPY statements) you perform by adding the COMMIT statement in the text box after the transaction statements. If you do not do so, the transaction rolls back.

In the following example, to insert values into table1, include a COMMIT statement in the text box and execute the two statements together:

```
INSERT INTO table1 VALUES (1,2);  
COMMIT;
```



Format

To input a series of queries, delimit them with a semicolon (;).

To automatically format the SQL text you have input, click the Format icon (</>).

Privileges

It is important when running queries in MC that the database administrator has correctly set up MC user privileges. The administrator must map all MC user profiles to their corresponding database user.

The Query Runner only permits MC users to perform actions that their corresponding Vertica database roles allow.

To set up user mappings, go to Home > MC Settings > User Management.

For more about how mapping MC user profiles to database users works, see [Granting Database Access to MC Users](#). For information about database-level users and privileges, see the [Managing Users and Privileges](#) section of the documentation.

Execute a Query

The Query Runner provides several ways to input a query to run:

- **Input text.** Enter the text for a query or series of queries into the text box.
- **Import a SQL script.** Click the Upload icon () to the top right of the text box to upload a SQL script (plain text file, typically with an extension of .sql). The queries from that file appears in the text box.
- **Enter a previous query from the Query History tab.** The Query History tab, on the left side of the page, displays the last 100 queries you have executed using the Query Runner on your current device and browser. Click any previous query in this tab to enter that query into the text box.

Hover over a query in the Query History tab to view all the query text. To clear queries from your history, hover over an individual query and click **x**, or click **Clear all** at the top of the tab. Click the star to the left of any query to favorite it, so it won't be cleared when you click **Clear all**.

Click **Execute Query** to run the queries you have input.

You can also execute only a portion of the text entered into the text box, as long as the selected text is a valid query. To do so, select that portion of the text. The **Execute selected text as query** button then appears below the text box.

For example, you might execute only a part of the entered text if you have uploaded a SQL script that containing multiple queries, but you decide to run only one of those queries.

To customize your execution settings, click the Settings icon () at the top right of the text box:

- **Row Limit:** Set the maximum number of rows to return. By default, the limit is 10000 rows.
- **Search Path:** Specify the schema to query.

```
1 SELECT sales_quantity, sales_dollar_amount, transaction_type, cc_name
2 FROM online_sales.online_sales_fact
3 INNER JOIN online_sales.call_center_dimension
4 ON (online_sales.online_sales_fact.call_center_key
5     = online_sales.call_center_dimension.call_center_key
6     AND sale_date_key = 156)
7 ORDER BY sales_dollar_amount DESC;
8 SELECT order_number, date_ordered
9 FROM store.store_orders_fact orders
10 WHERE orders.store_key IN (
11     SELECT store_key
12     FROM store.store_dimension
13     WHERE store_state = 'MA')
14     AND orders.vendor_key NOT IN (
15     SELECT vendor_key
16     FROM public.vendor_dimension
17     WHERE vendor_state = 'MA')
18     AND date_ordered < '2012-03-01';
19 SELECT store_key, order_number, date_ordered
```

Execute Queries Execute selected text as query

Get Query Results

The Query Runner returns results in a table format. If you ran multiple queries simultaneously, the results window displays a tab for each set of results. View the number of rows returned and the query execution time at the bottom of the results window.

If your result returns many columns, you can click **Auto-resize all columns** in the top right of the results window for a better fit, or click and drag column borders to manually resize individual columns.

Sort results by clicking on a column name, or use the search bar to narrow down results.

Execute Queries

SELECT fat_cont SELECT sales_qu ↑ ↓

Query Results Query Plan Query Profile Export Data Auto-Resize all columns Search query results

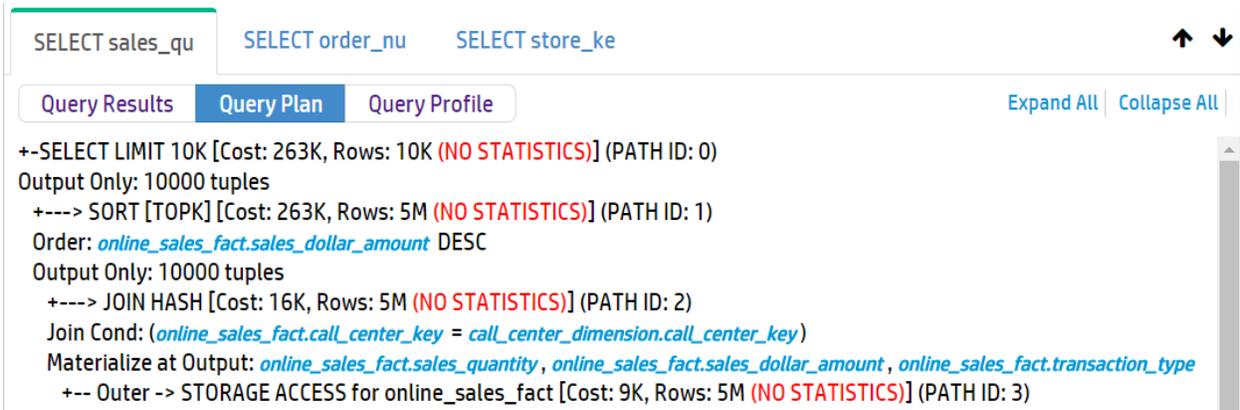
sales_quantity	sales_dollar_amount	transaction_type	cc_name
8	589	purchase	California
7	589	purchase	Central Midwest
8	589	purchase	South Midwest
1	587	purchase	New England
1	586	purchase	Other

2514 rows | Execution time: 0.176s

Query Plans and Profiles

Each query result also displays an option to retrieve the plan or profile for that query.

After retrieving a plan or profile, you can expand or collapse the results view to see different levels of detail. To view metadata for a projection or a column, click the object name in the path output. A pop-up window displays the metadata, if it is available.



The screenshot shows the Vertica Query Runner interface. At the top, there are three tabs: "SELECT sales_qu", "SELECT order_nu", and "SELECT store_ke". Below the tabs are three buttons: "Query Results", "Query Plan", and "Query Profile". To the right of these buttons are "Expand All" and "Collapse All" links. The main content area displays the following query plan:

```
+--SELECT LIMIT 10K [Cost: 263K, Rows: 10K (NO STATISTICS)] (PATH ID: 0)
Output Only: 10000 tuples
+----> SORT [TOPK] [Cost: 263K, Rows: 5M (NO STATISTICS)] (PATH ID: 1)
Order: online\_sales\_fact.sales\_dollar\_amount DESC
Output Only: 10000 tuples
+----> JOIN HASH [Cost: 16K, Rows: 5M (NO STATISTICS)] (PATH ID: 2)
Join Cond: (online\_sales\_fact.call\_center\_key = call\_center\_dimension.call\_center\_key)
Materialize at Output: online\_sales\_fact.sales\_quantity, online\_sales\_fact.sales\_dollar\_amount, online\_sales\_fact.transaction\_type
+-- Outer -> STORAGE ACCESS for online\_sales\_fact [Cost: 9K, Rows: 5M (NO STATISTICS)] (PATH ID: 3)
```

Note that the Query Runner does not automatically provide query profiles for queries that run for less than 1 second. To do so, prepend the word PROFILE to the query and run it.

You can also profile your query on the **Query Plan** page. The Query Plan page provides more details about both plan and profile results, including a query plan drilldown by node, a tree path view, and a profile analysis.

Keyboard Shortcuts



The Query Runner provides the following keyboard shortcuts:

- **?**: Press the question mark to display or dismiss a list of the available keyboard shortcuts. (You can also click the question mark icon at the top right of the text box to view this list.)
- **alt + ↑**: Press alt + up arrow to decrease the height of the text box.
- **alt + ↓**: Press alt + down arrow to increase the height of the text box.
- **ctrl + enter**: Press ctrl + enter to run the query.
- **ctrl + shift + enter**: Press ctrl + shift + enter to run selected text.

See Also

- [Granting Database Access to MC Users](#)
- [Managing Users and Privileges](#)
- [Queries](#)
- [SQL Reference Manual](#)
- [Using vsql](#)

Managing Queries in MC

Management Console can show you a query plan in easy-to-read format, where you can review the optimizer's strategy for executing a specific query. You can view a query plan in either of two ways:

- View the plan of an active query.
- View the plan for any query that you manually specify.

Access the Plan of an Active Query

1. At the bottom of the Management Console window, click the **Activity** tab.
2. From the list at the top of the page, select **Queries**.
3. On the activity graph, click the data point that corresponds to the query you want to view.
4. In the View Plan column, click **Explain** next to the command for which you want to view the query plan. Only certain queries use query plans—for example, SELECT, INSERT, DELETE, and UPDATE.
5. In the Explain Plan window, click **Explain**. Vertica generates the query plan.
6. (Optional) View the output in Path Information view or Tree Path view. To do so, click the respective view buttons on the left of the output box.

Access the Plan for a Specific Query

1. Locate the query for which you want to see the query plan in either of the following ways:
 - **Queries Not Running** — In the Explain window, type or paste the query text into the text box.
 - **Queries Currently Running** — In the Find a Query By ID input window, perform one of the following actions:

- Enter the query statement and transaction ID.
- Click the **Browse Running Queries** link.

Caution: Entering the word EXPLAIN before the query results in a syntax error.

2. Click **Explain**. Verticagenerates the plan.

If the query is invalid, Management Console highlights in red the parts of your query that might have caused a syntax error.

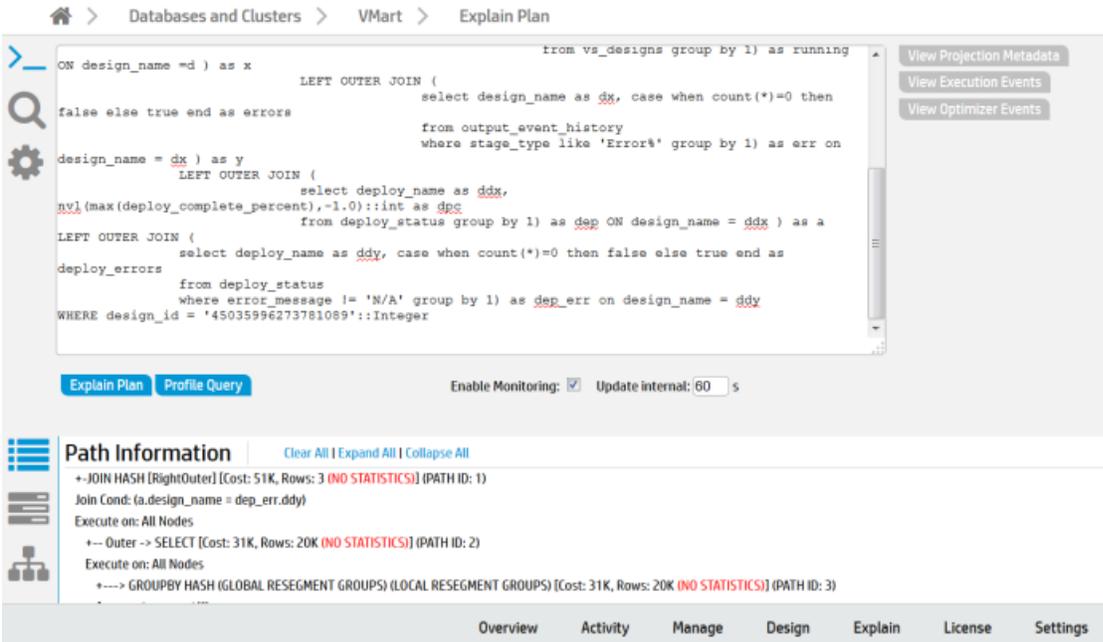
3. (Optional) View the output in Path Information view or Tree Path view. To do so, click the respective view buttons on the left of the output box.

Accessing Query Plans in Management Console

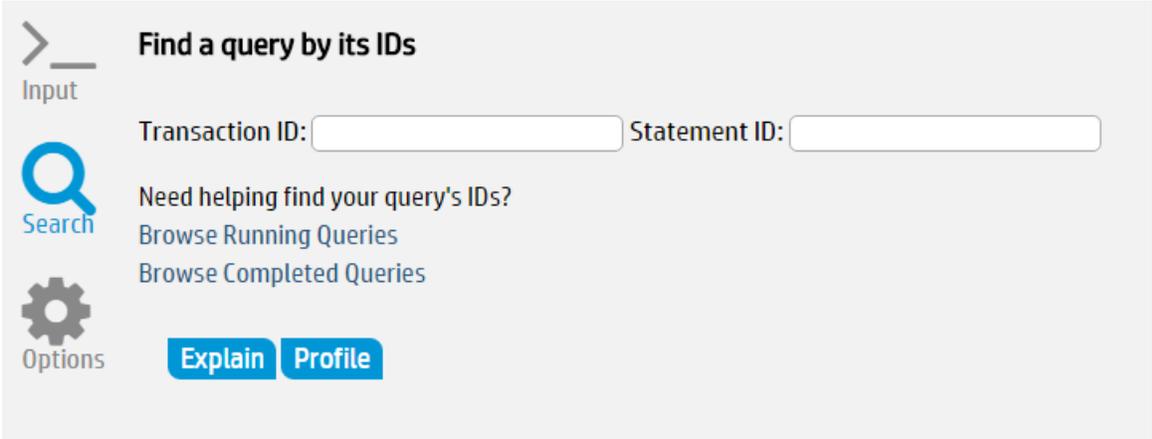
You can access query plans in Management Console in two ways:

- In the Detail page for query-related charts on the database's Activity page, click **Explain** next to a query to view a plan for that query.
- Enter a query manually on the Explain page and click **Explain Plan**.

In both cases, the following window opens:



You can also enter the transaction ID and statement ID or browse running or completed queries in the Find a Query input window:



In the output window, you can perform the following tasks related to the query you entered:

- [Expand and collapse query paths.](#)
- [Clearing query data.](#)
- [View projection and column metadata.](#)
- [Use different query plan views.](#)

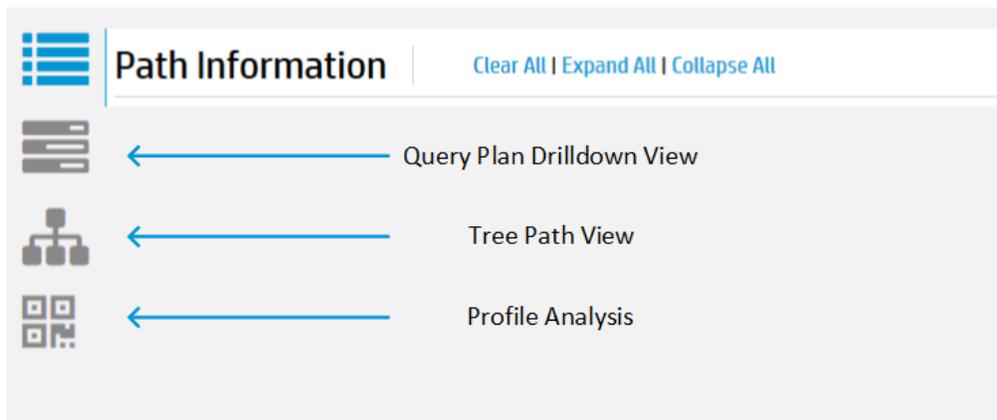
Query Plan View Options

Vertica Management Console provides two views for displaying query plans:

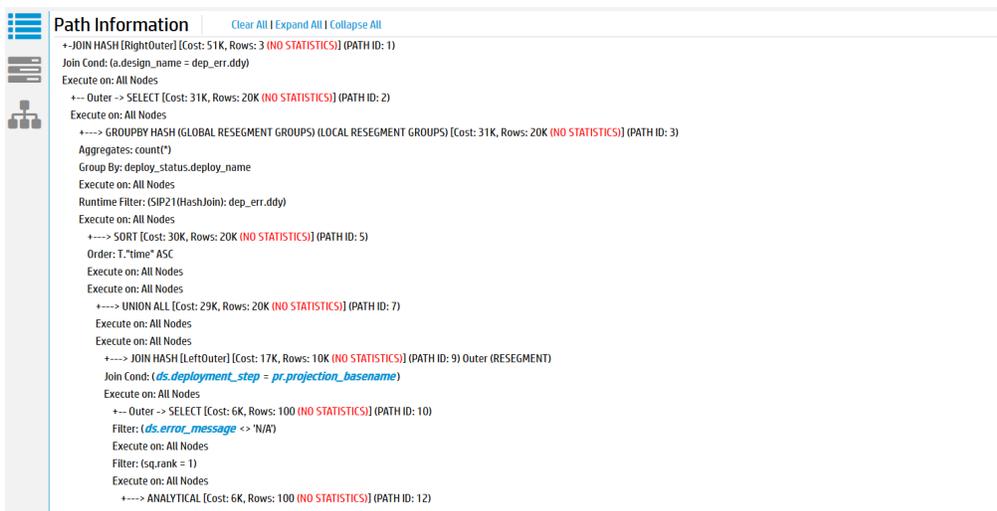
- Path Information
- Tree Path

Note: Query Plan Drilldown and Profile Analysis output views are only available when you run [PROFILE](#).

You can change the query plan view using the icons on the bottom portion of the **Explain** page.

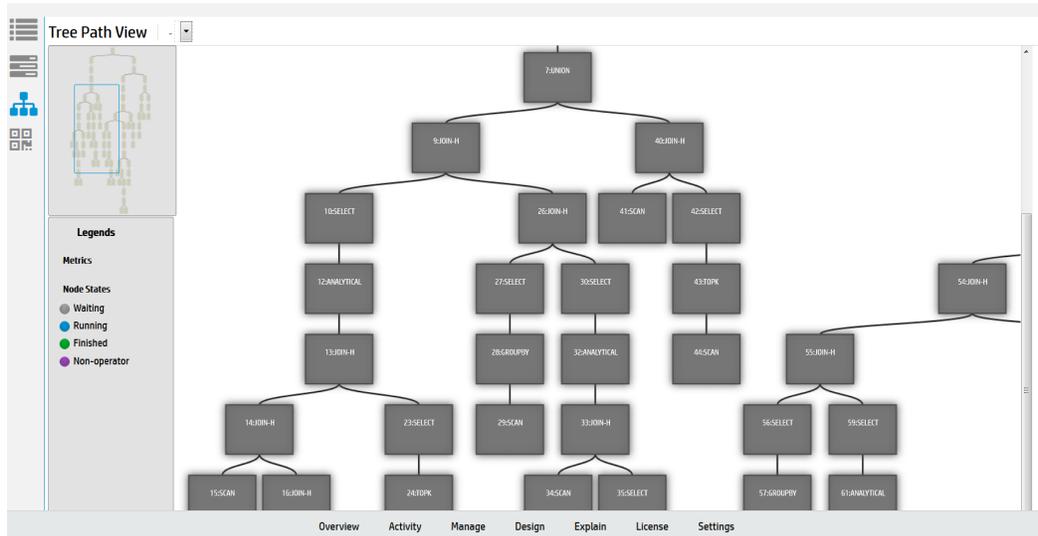


The **Path Information** view displays the query plan path. You can expand or collapse the view to see different levels of detail. To view metadata for a projection or a column, click the object name in the path output. A pop-up window displays the metadata, if it is available.



The

The **Tree Path** view details the query plan in the form of a tree. When you run EXPLAIN, the tree view does not contain any metrics because the query has not yet executed.



Expanding and Collapsing Query Paths

The EXPLAIN window initially displays the full query plan as generated by the [EXPLAIN](#) command. Query plans can be lengthy, so you might want to modify the display so you can focus only on areas of interest:

- Collapse All collapses all query paths, and displays only a summary of each path.
- Expand All expands all query paths.
- Click the first line of a path to display details for that path. To collapse that path, click its first line again.

For details about EXPLAIN command output, see [EXPLAIN-Generated Query Plans](#).

Clearing Query Data

After you finish reviewing the current query data, click Clear All to clear the query text and data. Alternatively, to display information about another query, enter the query text and click

Explain or Profile.

Viewing Projection and Column Metadata

In the Management Console EXPLAIN window, when query paths are expanded in the Path Information view, Projection lines contain a projection name and Materialize lines contain one or more column names.

To view metadata for a projection or a column, click the object name. A pop-up window displays the metadata. The following image on the left shows example projection metadata and the image on the right shows example column metadata.

Note: Most system tables do not have metadata.

Details [x]	
name	online_sales_fac
id	450359962765!
owner	uidbadmin
schema	online_sales
node	null
anchor_table	online_sales_fac
isprejoin	false
create_type	DESIGNER
verified_fault_tolerance	0
isuptodate	true
hasstatistics	true
issegmented	true
issuperprojection	true

Details [x]	
name	cc_name
max	Southwest
position	3
min	California
ndv	11
sort_order	4

When you are done viewing the metadata, close the pop-up window.

Creating a Database Design in Management Console

[Database Designer](#) creates a design that provides excellent performance for ad-hoc queries and specific queries while using disk space efficiently. Database Designer analyzes the logical schema definition, sample data, and sample queries, and creates a physical schema that you can deploy.

For more about how Database Designer works, see the [Creating a Database Design](#) section of the documentation, and [About Database Designer](#).

To use Management Console to create an optimized design for your database, you must be a DBADMIN user or have been assigned the DBDUSER role.

Management Console provides two ways to create a design:

- **Wizard**—This option walks you through the process of configuring a new design. Click **Back** and **Next** to navigate through the Wizard steps, or **Cancel** to cancel creating a new design.

To learn how to use the Wizard to create a design, see [Using the Wizard to Create a Design](#).

- **Manual**—This option creates and saves a design with the default parameters.

To learn how to create a design manually, see [Creating a Design Manually](#).

Tip: If you have many design tables that you want Database Designer to consider, it might be easier to use the Wizard to create your design. In the Wizard, you can submit all the tables in a schema at once; creating a design manually requires that you submit the design tables one at a time.

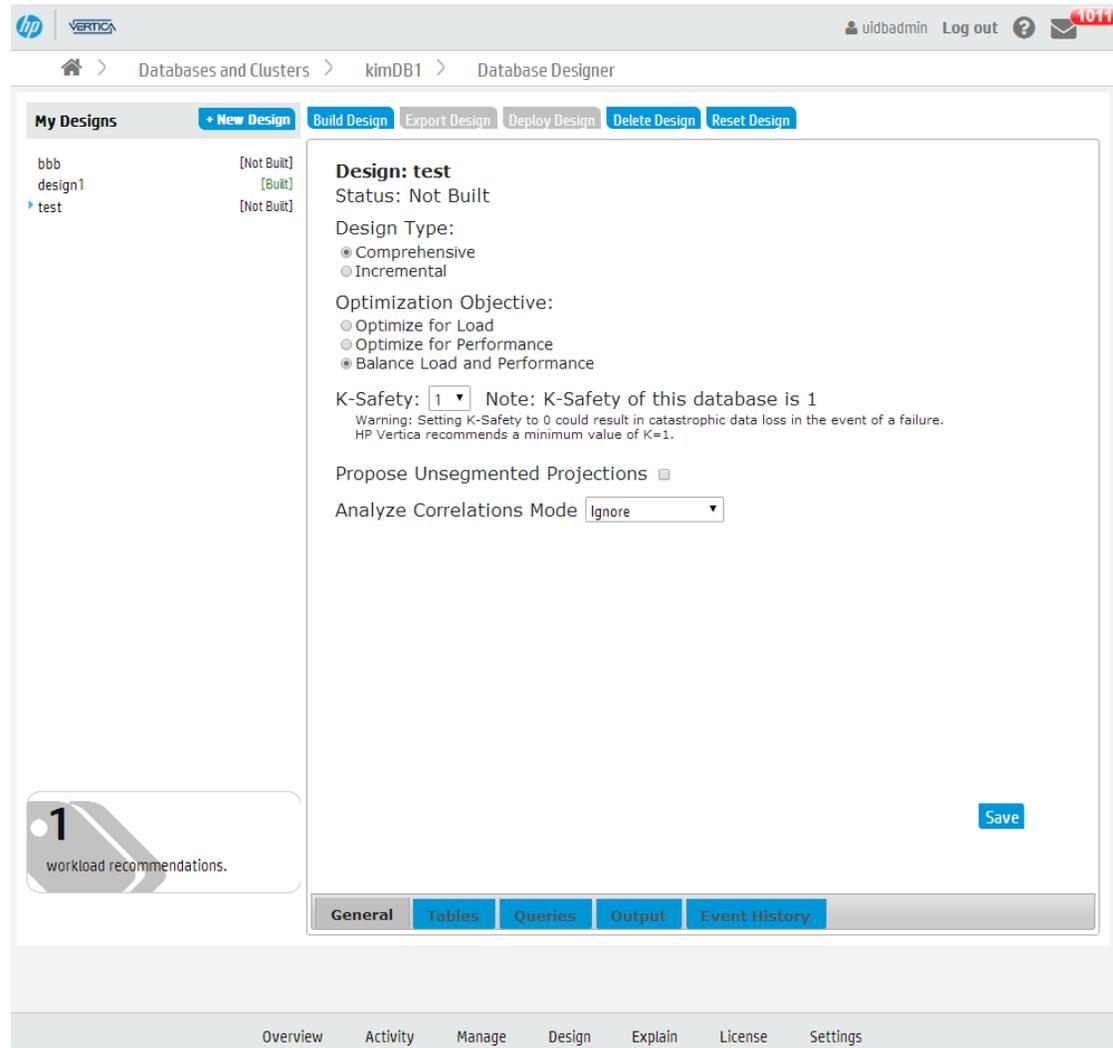
Using the Wizard to Create a Design

Take these steps to create a design using the Management Console's Wizard:

1. On your database's dashboard, click the **Design** tab at the bottom of the page to navigate to the Database Designer page.

The left side of the Database Designer page lists the database designs you own, with the most recent design you worked on highlighted. That pane also lists the current status of the design. Details about the most recent design appear in the main pane.

The main pane contains details about the selected design.



2. To create a new design, click **New Design**.
3. Enter a name for your design, and click **Wizard**.

For more information, see [Design Name](#).

4. Navigate through the Wizard using the **Back** and **Next** buttons.

5. To build the design immediately after exiting the Wizard, on the **Execution Options** window, select **Auto-build**.

Important: Micro Focus does not recommend that you auto-deploy the design from the Wizard. There may be a delay in adding the queries to the design, so if the design is deployed but the queries have not yet loaded, deployment may fail. If this happens, reset the design, check the **Queries** tab to make sure the queries have been loaded, and deploy the design.

6. When you have entered all the information, the Wizard displays a summary of your choices. Click **Submit Design** to build your design.

Summary

Review these design choices. Click Submit Design when you finish configuring your design.

Design Name: VMart_design
Design Type: comprehensive
Optimization Objective: balanced
Schemas: public, store, online_sales
K-Safety: 1
Analyze Correlation Mode: Ignore
Propose Unsegmented Projections: true
Query File to Upload:
User Query Repository: true
Execution Options - Analyze Statistics: true
Execution Options - Auto-build: true
Execution Options - Auto-deploy: false

[Back](#) [Submit Design](#) [Cancel](#)

See Also

- [About Database Designer](#)
- [Creating a Design Manually](#)

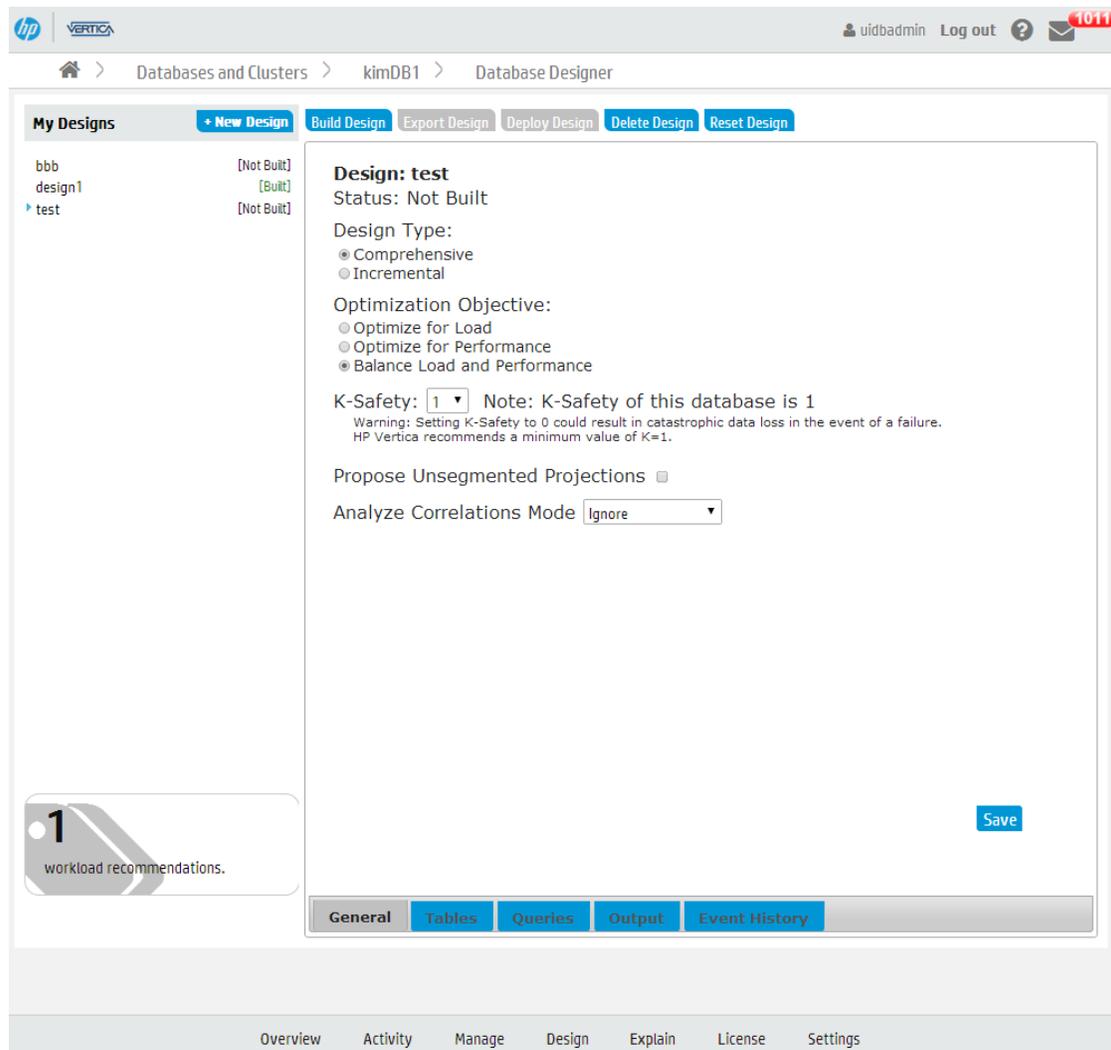
Creating a Design Manually

To create a design using Management Console and specify the configuration, take these steps.

1. On your database's dashboard, click the **Design** tab at the bottom of the page to navigate to the Database Designer page.

The left side of the Database Designer page lists the database designs you own, with the most recent design you worked on highlighted. That pane also lists the current status of the design. Details about the most recent design appear in the main pane.

The main pane contains details about the selected design.



2. To create a new design, click **New Design**.
3. Enter a name for your design and select **Manual**.

The main **Database Design** window opens, displaying the default design parameters. Vertica has created and saved a design with the name you specified, and assigned it the default parameters.

For more information, see [Design Name](#).

4. On the **General** window, modify the design type, optimization objectives, K-safety, Analyze Correlations Mode, and the setting that allows Database Designer to create unsegmented projections.

If you choose **Incremental**, the design automatically optimizes for the desired queries, and the K-safety defaults to the value of the cluster K-safety; you cannot change these values for an incremental design.

Analyze Correlations Mode determines if Database Designer analyzes and considers column correlations when creating the design. For more information, see [DESIGNER_SET_ANALYZE_CORRELATIONS_MODE](#).

5. Click the **Tables** tab. You must submit tables to your design.
6. To add tables of sample data to your design, click **Add Tables**. A list of available tables appears; select the tables you want and click **Save**. If you want to remove tables from your design, click the tables you want to remove, and click **Remove Selected**.

If a design table has been dropped from the database, a red circle with a white exclamation point appears next to the table name. Before you can build or deploy the design, you must remove any dropped tables from the design. To do this, select the dropped tables and click **Remove Selected**. You cannot build or deploy a design if any of the design tables have been dropped.

7. Click the **Queries** tab. To add queries to your design, do one of the following:
 - To add queries from the [QUERY_REQUESTS](#) system table, click **Query Repository**. In the Queries Repository dialog that appears, you can sort queries by most recent, most frequently executed, and longest running. Select the desired queries and click **Save**. All valid queries that you selected appear in the **Queries** window.
 - To add queries from a file, select **Upload**. All valid queries in the file that you select are added to the design and appear in the **Queries** window. (This option is only available for the [MC super administrator role](#).)

Database Designer checks the validity of the queries when you add the queries to the design and again when you build the design. If it finds invalid queries, it ignores them.

If you have a large number of queries, it may take time to load them. Make sure that all the queries you want Database Designer to consider when creating the design are listed in the **Queries** window.

8. Once you have specified all the parameters for your design, you should build the design. To do this, select your design and click **Build Design**.

9. Select **Analyze Statistics** if you want Database Designer to analyze the statistics before building the design.

For more information see [Statistics Analysis](#).

10. If you do not need to review the design before deploying it, select **Deploy Immediately**. Otherwise, leave that option unselected.
11. Click **Start**. On the left-hand pane, the status of your design displays as **Building** until it is complete.
12. To follow the progress of a build, click **Event History**. Status messages appear in this window and you can see the current phase of the build operation. The information in the Event History tab contains data from the `OUTPUT_EVENT_HISTORY` system table.
13. When the build completes, the left-hand pane displays **Built**. To view the deployment script, select your design and click **Output**.
14. After you deploy the design using Management Console, the deployment script is deleted. To keep a permanent copy of the deployment script, copy and paste the SQL commands from the **Output** window to a file.
15. Once you have reviewed your design and are ready to deploy it, select the design and click **Deploy Design**.
16. To follow the progress of the deployment, click **Event History**. Status messages appear in this window and you can see the current phase of the deployment operation.

In the Event History window, while the design is running, you can do one of the following:

- Click the blue button next to the design name to refresh the event history listing.
 - Click **Cancel Design Run** to cancel the design in progress.
 - Click **Force Delete Design** to cancel and delete the design in progress.
17. When the deployment completes, the left-hand pane displays **Deployment Completed**. To view the deployment script, select your design and click **Output**.

Your database is now optimized according to the parameters you set.

Getting Tuning Recommendations in MC

If queries perform sub-optimally, you can get tuning recommendations for them, as well as for hints about optimizing database objects, by using the Workload Analyzer (WLA).

WLA is a Vertica utility that analyzes system information in Vertica system tables. It then returns a set of tuning recommendations based on statistics, system and data collector events, and database/table/projection design. You can use these recommendations to tune query performance.

View WLA Recommendations

WLA recommendations are available through the database's **Overview** page.

1. On the right side of the Overview page in the Quick Stats bar, view the Workload Analyzer module. This module alerts you to the number of recommendations WLA has generated.



2. Click the number to view the WLA recommendations on the Database Designer page. The Workload Analyzer Results dialog displays tuning recommendations and the resource cost of running each command (Low, Medium, or High).

For additional information about tuning recommendations and cost see [Analyzing Workloads](#) in the Administrator's Guide and [ANALYZE_WORKLOAD](#) in the SQL Reference Manual.

The following image shows a list of WLA tuning recommendations for a database. In the output, WLA advises that you run the Database Designer on several tables and provides the cost estimate of running these operations.

Note that the cost of running Database Designer is high. When you see a high cost, you might want to run the recommended tuning action after hours.

Tuning recommendation	Cost
run database designer on table online_sales.call_center_dimension	HIGH
run database designer on table online_sales.online_page_dimension	HIGH
run database designer on table online_sales.online_sales_fact	HIGH
run database designer on table public.casey	HIGH
run database designer on table public.date_dimension	HIGH
run database designer on table public.employee_dimension	HIGH
run database designer on table public.helpful_button_stage	HIGH
run database designer on table public.inventory_fact	HIGH
run database designer on table public.my_table	HIGH
run database designer on table public.num_test	HIGH
run database designer on table public.num_test2	HIGH
run database designer on table public.numeric_table	HIGH
run database designer on table public.product_dimension	HIGH
run database designer on table public.promotion_dimension	HIGH
run database designer on table public.sample_table	HIGH

MC displays fifteen recommendations per page and includes a message at bottom left, which lets you know where you are within the total number of recommendations; for example, "Showing *n-nn* of *nn* entries." Depending on the total number of recommendations, you can move forward and backward through them by clicking the arrows at bottom right.

Tip: You can force the WLA task to run immediately by clicking **Update Recommendations** over the Cost column.

Set and Disable WLA Events

On MC, WLA automatically begins monitoring data one minute after the MC process starts. WLA then runs once per day, or immediately after you import a database to MC. It continually gathers data in the background as long as the database is running. If you haven't created a database yet, or if the database is down, WLA does nothing until the database is back up.

By default, WLA runs each day at 2 AM. To optimize when WLA uses resources, you can set WLA to run at a different time for any or all databases that MC monitors. You can also set MC to never run WLA automatically.

1. On the Home page, click **MC Settings**.
2. Click the **Monitoring** tab.
3. Under the **Workload Analyzer Assistant** section of the Monitoring page, select your time zone.
4. Select the radio button for one of the options:
 - **All Databases:** Select a time from the list. WLA will run at that time on all databases that MC monitors.
 - **Specific Database at Specific Time:** Select a database and a time from the list. At the time you specify, WLA will run at that time on the database you selected.
 - **Do Not Run Workload Analyzer On Any Database:** MC will never run WLA automatically on any database it monitors.
5. Click **Apply** at the top right of the page.

For additional information about tuning recommendations and their triggering event, see [Understanding WLA Triggering Conditions](#).

See Also

[Analyzing Workloads](#)

[Getting Tuning Recommendations](#)

Monitoring Using MC

Management Console gathers and retains history of important system activities about your MC-managed database cluster, such as performance and resource utilization. You can use MC charts to locate performance bottlenecks on a particular node, to identify potential improvements to Vertica configuration, and as a reference for what actions users have taken on the MC interface.

Note: MC directly queries Data Collector tables on the MC-monitored databases themselves. See [Management Console Architecture](#) in Vertica Concepts. For how to set up MC to query an alternative database for monitoring data, see [Extended Monitoring](#).

The following list describes some of the areas you can monitor and troubleshoot through the MC interface:

- Multiple database cluster states and key performance indicators that report on the cluster's overall health
- Information on individual cluster nodes specific to resources
- Database activity in relation to CPU/memory, networking, and disk I/O usage
- Query concurrency and internal/user sessions that report on important events in time
- Cluster-wide messages
- Database and agent log entries
- MC user activity (what users are doing while logged in to MC)
- Issues related to the MC process
- Error handling and feedback

About Chart Updates

MC charts update dynamically with text, color, and messages Management Console receives from the agents on the database cluster. This information can help you quickly resolve problems.

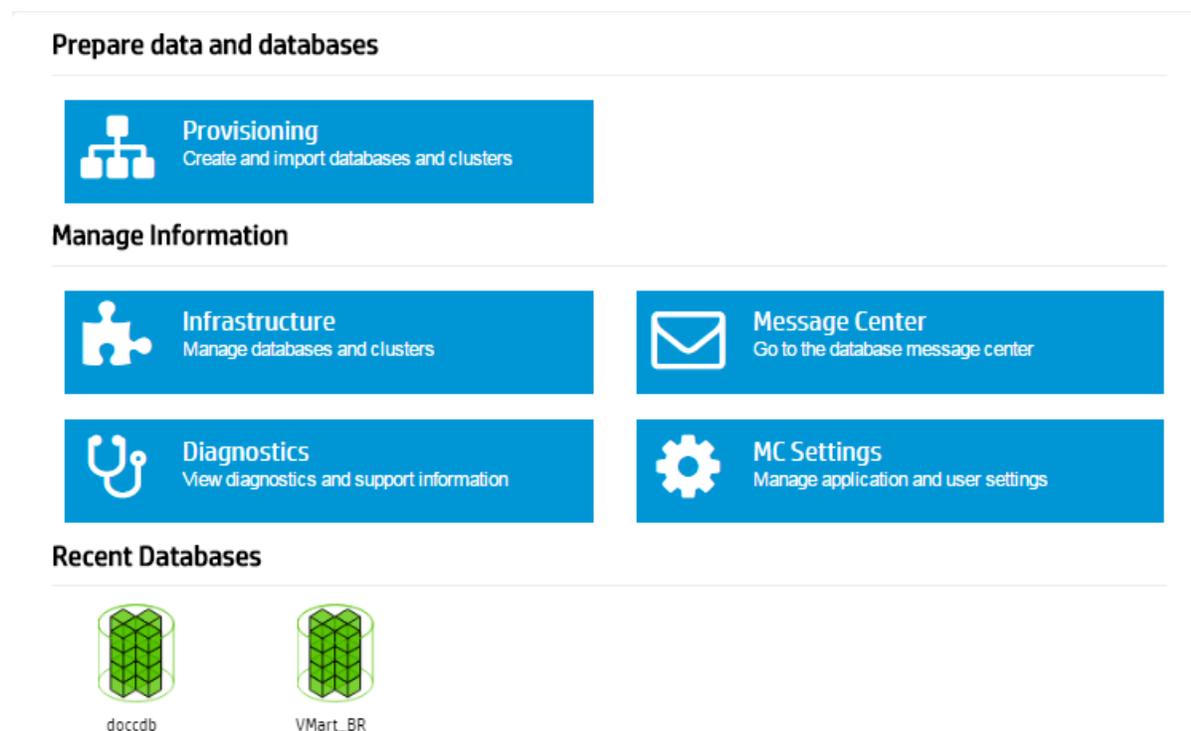
Each client session to MC uses a connection from `MaxClientSessions`, a database configuration parameter. This parameter determines the maximum number of sessions that

can run on a single database cluster node. Sometimes multiple MC users, mapped to the same database account, are concurrently monitoring the Overview and Activity pages. In such cases, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for `MaxClientSessions` on an MC-monitored database to account for extra sessions. See [Managing Sessions](#) for details.

Viewing the Home Page

After you [connect to MC](#) and sign in, the Home page displays. This page is the entry point to all Vertica database clusters and users managed by MC. Information on this page, as well as throughout the MC interface, will appear or be hidden, based on the permissions ([access levels](#)) of the user who is logged in. The following image is what an MC super administrator sees.



Tasks

Operations you can perform in Management Console are grouped into the following task-based areas:

- **Provision Databases.** Create and import databases, and connect this includes creating new empty databases and importing existing database clusters into Management Console. You can also set up an Ambari connection to import a Vertica cluster that resides in a Hadoop environment. See [Managing Database Clusters](#).
- **Existing Infrastructure.** View all the clusters and databases monitored by MC, stop and remove databases, and view details about databases and clusters. See [Managing Database Clusters](#).
- **MC Settings.** Configure MC and user settings, as well as use the MC interface to install Vertica on a cluster of hosts. See [Managing MC Settings](#).
- **Message Center.** View, sort, and search database messages and optionally export messages to a file. See [Monitoring Database Messages in MC](#).
- **Diagnostics.** View and resolve MC-related issues, as well as browse Vertica agent and audit logs. See [Troubleshooting with MC Diagnostics](#).

Recent Databases

The **Recent Databases** section displays all databases that you created on or imported into the MC interface. You can install one or more databases, on the same cluster or on different clusters, but you can have only one database running on a single cluster at a time. UP databases appear in green and DOWN databases are red. An empty space under Recent Databases means that you have not yet created or imported a database into the MC interface, or do not have permission to view any databases managed by MC.

Monitoring Same-Name Databases on MC

If you are monitoring two databases with identical names on different clusters, you can determine which database is associated with which cluster by clicking the database icon on MC's Databases and Clusters page to view its dialog box. Information in the dialog displays the cluster on which the selected database is associated.

Viewing the Overview Page

The Overview page displays a dynamic dashboard view of your database.

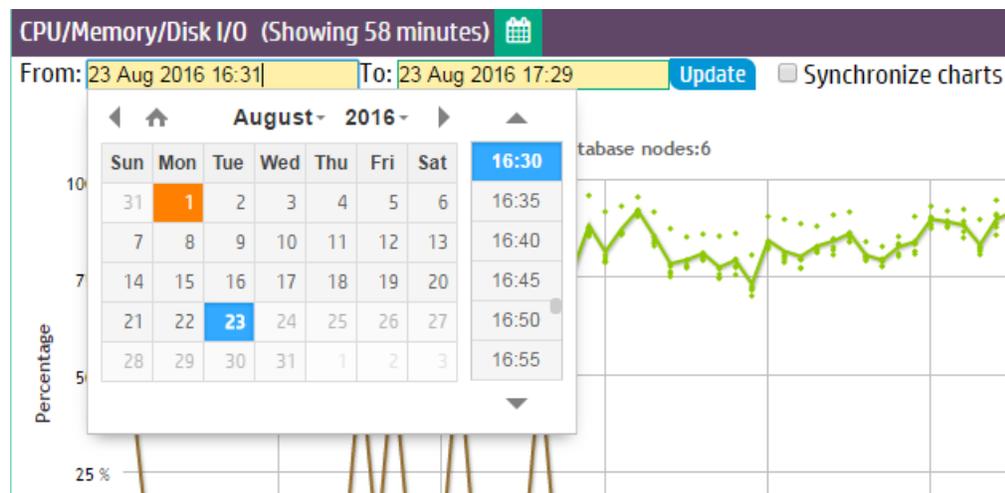
The page provides three tabs: Status Summary, System Health, and Query Synopsis. Each tab contains charts and filters displaying information about your cluster. The QuickStats widgets on the right of the page display alerts and statistics about the state of your cluster.

Information on this page updates every minute, but you can postpone updates by deselecting Auto Refresh Charts in the toolbar.

Chart Viewing Options

You can specify time frames for some charts, which display a calendar icon in their title bars. Click the calendar icon to specify the time frame for that module.

On the Status Summary tab, you can select **Synchronize charts** to simultaneously apply the specified time frame to all charts on that tab.



If you have enabled extended monitoring on your database, MC can display longer ranges of data in certain charts. See [Extended Monitoring](#). If a chart is using extended monitoring data, the rocket ship icon appears in the title bar:



You can expand some charts to view them in larger windows. Click the expand icon in the title bar to do so:



Quick Stats

The Quick Stats sidebar on the right of the page provides instant alerts and information about your cluster's status.

Database Nodes Health

	0		0		0		6	Total: 6 Version: 8.0.0-0
Down		Critical		Recovering		Up		

Running And Queued Queries

	29		0
Running		Queued	

Projections

	18		0		0	For Largest Schema: public
Total		Unsegmented		Unsafe/Not Up To Date		

Disk Space Usage

	0	Total: 6 Nodes Based On Config Param For Alerts On Disk Usage: >60%
Low		

Workload Analyzer

	33
Recommendations	

I/O Wait Notices

	6		6	Threshold value exceeded within last hour
CPU		Network		
Nodes with CPU wait time exceeding: 1 sec		Nodes with network errors exceeding: 0%		

License Consumption

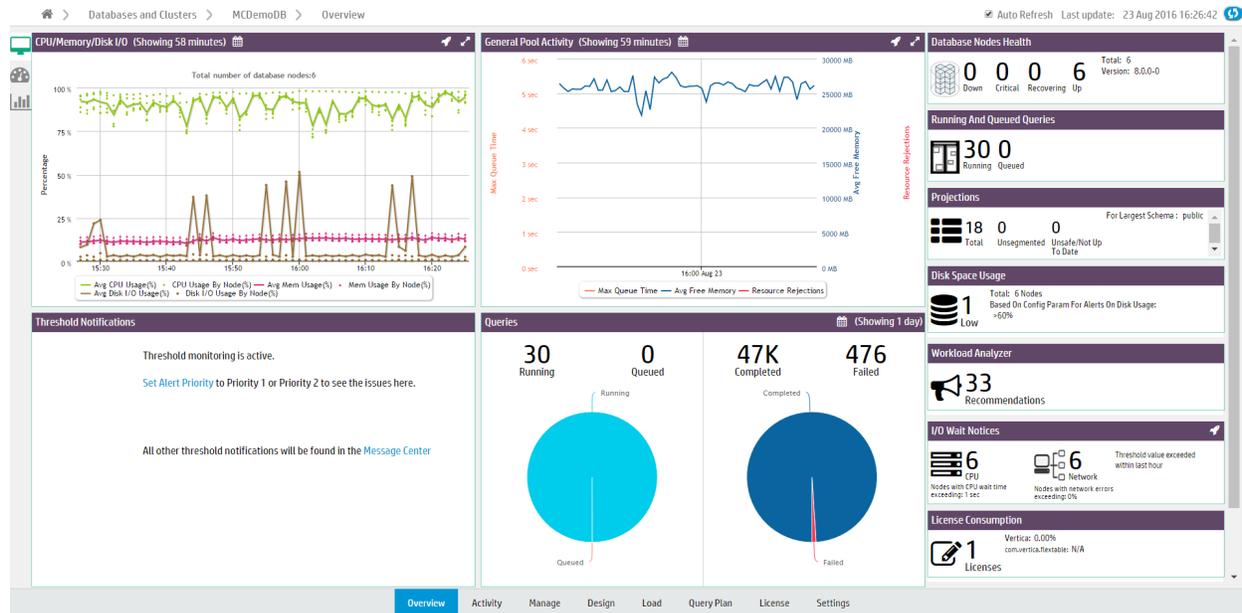
	1	Vertica: 0.00% com.vertica.flextable: N/A
Licenses		

Unread Messages (This Week)

	6	Total Unread Messages: 411
Unread High Priority		

- **Database Nodes Health** displays which nodes are down, critical, recovering, or up. Critical and recovering nodes are included in the total nodes considered "up" by the database. Click a node value to open the Manage page.
- **Running and Queued Queries** displays current queries in the database. Click the query values to open the Query Monitoring charts.
- **Projections** displays the number of total projections, unsegmented projections, and unsafe projections for the database schema with the most projections. Click a value to open the Table Treemap chart.
- **Disk Space Usage** alerts you to the number of nodes that are low on disk space. Click the value to go to the Manage page. On the Manage page, the Storage Used KPI View is displayed.
- **Workload Analyzer (WLA)** analyzes system information retained in [SQL system tables](#) and provides tuning recommendations, along with the cost (low, medium, or high) of running the command. See [Analyzing Workloads](#) for more information.
- **I/O Wait Notices** displays the number of nodes that, in the last hour, have recorded Disk I/O waits and Network I/O waits exceeding the wait threshold (1 second for Disk and 0 seconds for Network).
- **License Consumption** displays the number of licenses your database uses, and the percentage of your Vertica Community Edition or Premium Edition license being used.
- **Unread Messages** display the number of unread messages and alerts for the database. This count differs from the number of total messages across all your databases. Click the value to open the Message Center.

Status Summary



The Status Summary tab displays four modules that provide a general overview of the status of your cluster:

- The **CPU/Memory/Disk I/O Usage** module shows cluster resource usage. The chart displays the number of nodes in the database cluster and plots average and per-node percentages for CPU, memory, and disk I/O usage. Select a resource type from the legend to remove or add it from the chart display.

Click a data point (which represents a node) to open the Manage page. See [Monitoring Cluster CPU/Memory](#).

- The **General Pool Activity** module displays GENERAL pool activity. The chart displays average query queue times, average GENERAL pool free memory, and resource rejections. Use this chart to see how much free memory there is in GENERAL pool, or if there have been high queue times. Click the expand icon in the title bar to open the chart in a bigger window.

Click a data point to open the [Resource Pools Monitoring](#) chart. See [Managing Workloads](#).

- The **Thresholds Notifications** module displays alerts generated when a threshold has been exceeded in the database. Notifications are categorized by System Health and Performance.

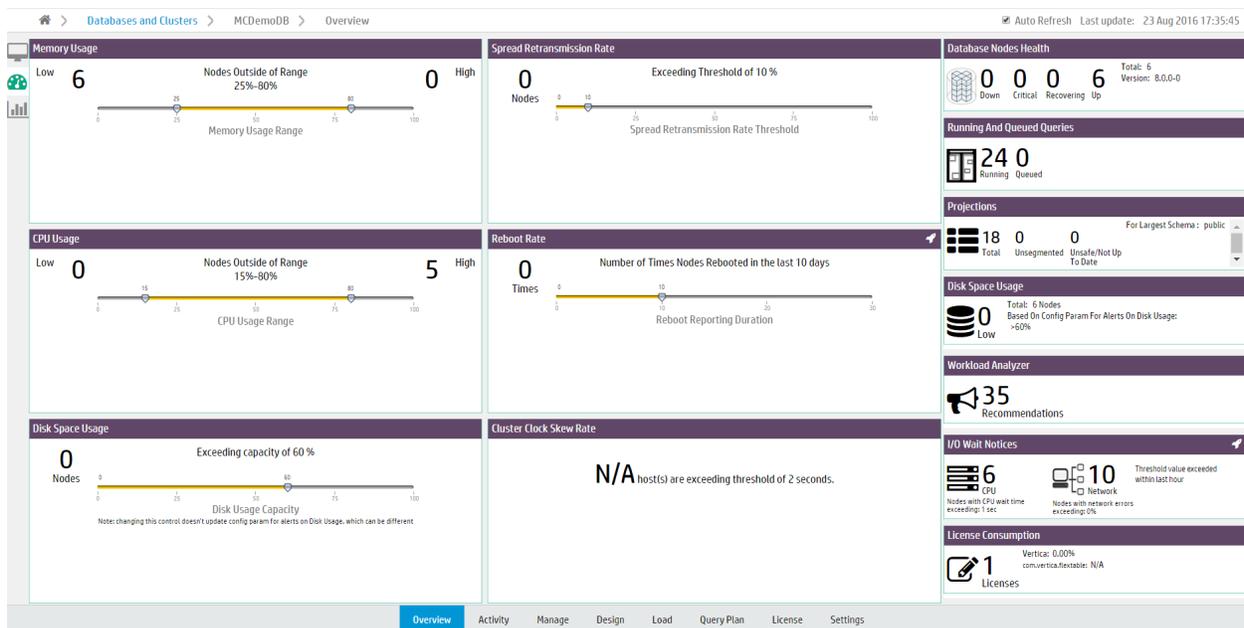
In the module, you can acknowledge an alert (which marks it as read) or click the X to stop monitoring that threshold (which stops you receiving similar alerts in the future).

Customize thresholds and alert priorities for these notifications in the Thresholds tab of the database Settings page. See [Customizing Threshold-Based Notifications](#).

- The **Queries** module displays query statistics. The first pie chart displays running and queued queries in the last 24 hours. The second chart displays completed and failed queries for the time frame you specify.

Click a query count number above the chart to open the Query Monitoring chart. See [Monitoring Running Queries](#).

System Health



The System Health tab provides a summary of your system resource usage and node information, with filters that allow you to view resource usage within the ranges you specify.

Note: Adjusting the filters on the System Health tab does not affect any database or MC settings.

- The **Memory Usage** filter displays the number of nodes with high and low memory usage. Move the sliders to adjust the memory usage range filter.

For example, if you specify a range of 25% to 75% memory usage, the filter displays how many nodes are using less than 25% of memory (Low) and how many are using more than 75% (High). Hover your cursor over the Low and High values to see lists of what nodes fall, respectively, below or above the memory usage range you specified.

Click a node value to go to the Manage page, which displays the Memory Utilization KPI View.

- The **Spread Retransmission Rate** filter displays the number of nodes with high spread retransmission rates. When a node's retransmission rate is too high, it is not communicating properly with other nodes. Move the slider to adjust the retransmission rate filter.

Hover your cursor over the Nodes value to see a list of what nodes exceeded the spread retransmission rate you specified. Click the node value to view spread retransmit rate alerts in the Message Center.

- The **CPU Usage** chart displays the number of nodes with high and low CPU usage. Move the sliders to adjust the CPU usage range filter. Hover your cursor over the Low and High values to see lists of what nodes are below or above range you specified.

Click a node value to go to the Manage page, which displays the CPU Utilization KPI View.

- The **Reboot Rate** filter displays the number of times nodes in the cluster have rebooted within the specified time frame. Use this filter to discover if nodes have gone down recently, or if there have been an unusual number of reboots. Move the slider to adjust the number of days. Hover over the Times value to see a list of the nodes that have rebooted and the times at which they did so.
- The **Disk Space Usage** filter displays the number of nodes with high disk space usage. Move the slider to adjust the disk usage filter. Hover your cursor over the Nodes value to see a list of what nodes exceed the acceptable range.

Click the nodes value to go to the Manage page, which displays the Storage Used KPI View.

- The **Cluster Clock Skew Rate** module displays the number of nodes that exceed a clock skew threshold. Nodes in a cluster whose clocks are not in sync can interfere with time-related database functions, the accuracy of database queries, and Management Console's monitoring of cluster activity.

Query Synopsis



The Query Synopsis page provides two modules that report system query activity and resource pool usage:

- The **Query Statistics** module displays four bar charts that provide an overview of running, queued queries, failed, and completed queries in the past 24 hours. Select one of the options at the top of the module to group the queries by **Resource Pools**, **Users**, or **Nodes**.

Click a bar on the chart to view details about those queries the [Query Monitoring](#) activity chart.

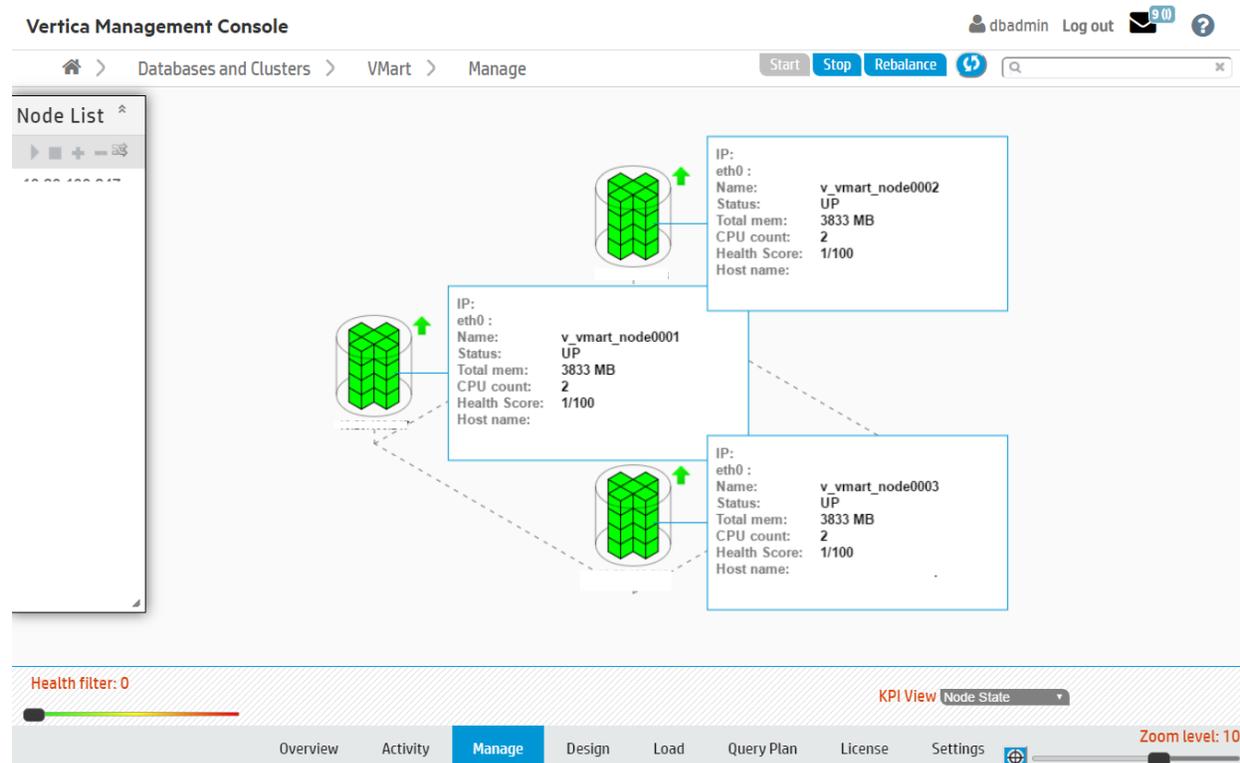
- The **User Query Type Distribution** chart provides an overview of user and system query activity. The chart reports the types of operation that ran. Hover your cursor over chart points for more details. Select a type of operation from the legend to remove or add it from the chart display. To zoom to a certain time frame, you can adjust the sliders at the bottom of the chart.

Click a bar in the graph to open the [Queries](#) chart.

Monitoring Cluster Nodes

For a visual overview of all cluster nodes, click the running database on the Databases and Clusters page and click the **Manage** tab at the bottom of the page to open the cluster status page.

The cluster status page displays the nodes in your cluster.



The appearance of the nodes indicate the following states:

- **Healthy:** The nodes appear green.
- **Up:** A small arrow to the right of the node points upward.
- **Critical:** The node appears yellow and displays a warning icon to the right.
- **Down:** The node appears red. To the right of the node, a red arrow points downwards.
- **Unplugged:** An orange outlet and plug icon appears to the right.

You can get information about a particular node by clicking it, an action that opens the [node details](#) page.

Filtering What You See

If you have a large cluster, where it might be difficult to view dozens to hundreds of nodes on the MC interface, you can filter what you see. The Zoom filter shows more or less detail on the cluster overall, and the Health Filter lets you view specific node activity; for example, you can slide the bar all the way to the right to show only nodes that are down. A message next to the health filter indicates how many nodes in the cluster are hidden from view.

On this page, you can perform the following actions on your database cluster:

- Add, remove and replace nodes
- Rebalance data across all nodes
- Stop or start (or restart) the database
- Refresh the view from information MC gathers from the production database
- View key performance indicators (KPI) on node state, CPU, memory, and storage utilization (see [Monitoring Cluster Performance](#) for details)

Note: Starting, stopping, adding, and dropping nodes and rebalancing data across nodes works with the same functionality and restrictions as those same tasks performed through the Administration Tools.

If You don't See What You Expect

If the cluster grid does not accurately reflect the current state of the database (for example if the MC interface shows a node in INITIALIZING state, but when you use the Administration Tools to View Database Cluster State, you see that all nodes are UP), click the Refresh button in the toolbar. Doing so forces MC to immediately synchronize with the agents and update MC with new data.

Don't press the F5 key, which redisplay the page using data from MC and ignores data from the agent. It can take several seconds for MC to enable all database action buttons.

Monitoring Cluster CPU/Memory

On the MC Overview page, the **CPU/Memory** subsection provides a graph-based overview of cluster resources during the last hour, which lets you quickly monitor resource distribution

across nodes.

This chart plots average and per-node percentages for both CPU and memory with updates every minute—unless you clear Auto Refresh Charts in the toolbar. You can also filter what the chart displays by clicking components in the legend at the bottom of the subsection to show/hide those components. Yellow data points represent individual nodes in the cluster at that point in time.

Investigating Areas of Concern

While viewing cluster resources, you might wonder why resources among nodes become skewed. To zoom in, use your mouse to drag around the problem area surrounding the time block of interest.

After you release the mouse, the chart refreshes to display a more detailed view of the selected area. If you hover your cursor over the node that looks like it's consuming the most resources, a dialog box summarizes that node's percent usage.

For more information, click a data point (node) on the graph to open MC's node details page. To return to the previous view, click **Reset zoom**.

See Also

- [Monitoring Node Activity](#)

Monitoring Cluster Performance

Key Performance Indicators (KPIs) are a type of performance measurement that let you quickly view the health of your database cluster through MC's **Manage** page. These metrics, which determine a node's color, make it easy for you to quickly identify problem nodes.

Metrics on the database are computed and averaged over the latest 30 seconds of activity and dynamically updated on the cluster grid.

How to Get Metrics on Your Cluster

To view metrics for a particular state, click the menu next to the **KPI View** label at the bottom of the Manage page, and select a state.

MC reports KPI scores for:

- **Node state**—(default view) shows node status (up, down, k-safety critical) by color; you can filter which nodes appear on the page by sliding the health filter from left to right
- **CPU Utilization**—average CPU utilization
- **Memory Utilization**—average percent RAM used
- **Storage Utilization**—average percent storage used

After you make a selection, there is a brief delay while MC transmits information back to the requesting client. You can also click **Sync** in the toolbar to force synchronization between MC and the client.

Node Colors and What They Mean

Nodes in the database cluster appear in color. Green is the most healthy and red is the least healthy, with varying color values in between.

Each node has an attached information dialog box that summarizes its score. It is the score's position within a range of 0 (healthiest) to 100 (least healthy) that determines the node's color *bias*. Color bias means that, depending on the value of the health score, the final color could be slightly biased; for example, a node with score 0 will be more green than a node with a score of 32, which is still within the green range but influenced by the next base color, which is yellow. Similarly, a node with a score of 80 appears as a dull shade of red, because it is influenced by orange.

MC computes scores for each node's color bias as follows:

- 0-33: green and shades of green
- 34-66: yellow and shades of yellow
- 67-100: red and shades of red shades

If the unhealthy node were to consume additional resources, its color would change from a dull orange-red to a brighter red.

Filtering Nodes From the View

The health filter is the slider in the lower left area of page. You can slide it left to right to show or hide nodes; for example, you might want to hide nodes with a score smaller than a certain value so the UI displays only the unhealthy nodes that require immediate attention. Wherever

you land on the health filter, an informational message appears to the right of the filter, indicating how many nodes are hidden from view.



Filtering is useful if you have many nodes and want to see only the ones that need attention, so you can quickly resolve issues on them.

Monitoring System Resources

MC's **Activity** page provides immediate visual insight into potential problem areas in your database's health by giving you graph-based views of query and user activity, hardware and memory impact, table and projection usage, system bottlenecks, and resource pool usage.

Select one of the following charts in the toolbar menu:

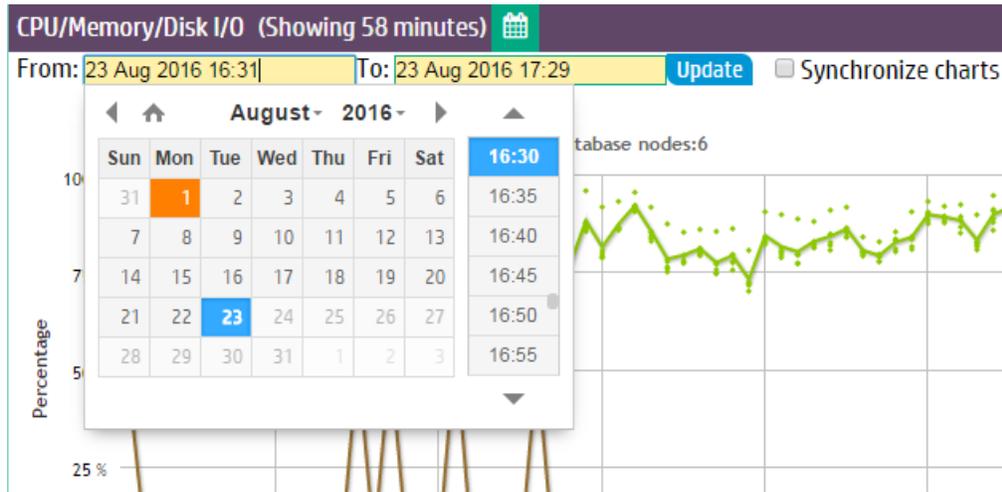
- [Queries](#)
- [Internal Sessions](#)
- [User Sessions](#)
- [Memory Usage](#)
- [System Bottlenecks](#)
- [User Query Phases](#)
- [Monitoring Table Utilization and Projections](#)
- [Query Monitoring](#)
- [Resource Pool Monitoring](#)
- [Monitoring Catalog Memory](#)

How up to date is the information?

System-level activity charts automatically update every five minutes, unless you clear Auto Refresh in the toolbar. Depending on your system, it could take several moments for the charts to display when you first access the page or change the kind of resource you want to view.

Chart Viewing Options

You can specify time frames for some charts, which display a calendar icon in their title bars. Click the calendar icon to specify the time frame for that module.



If you have enabled extended monitoring on your database, MC can display longer ranges of data in certain charts. See [Extended Monitoring](#). If a chart is using extended monitoring data, the rocket ship icon appears in the title bar:

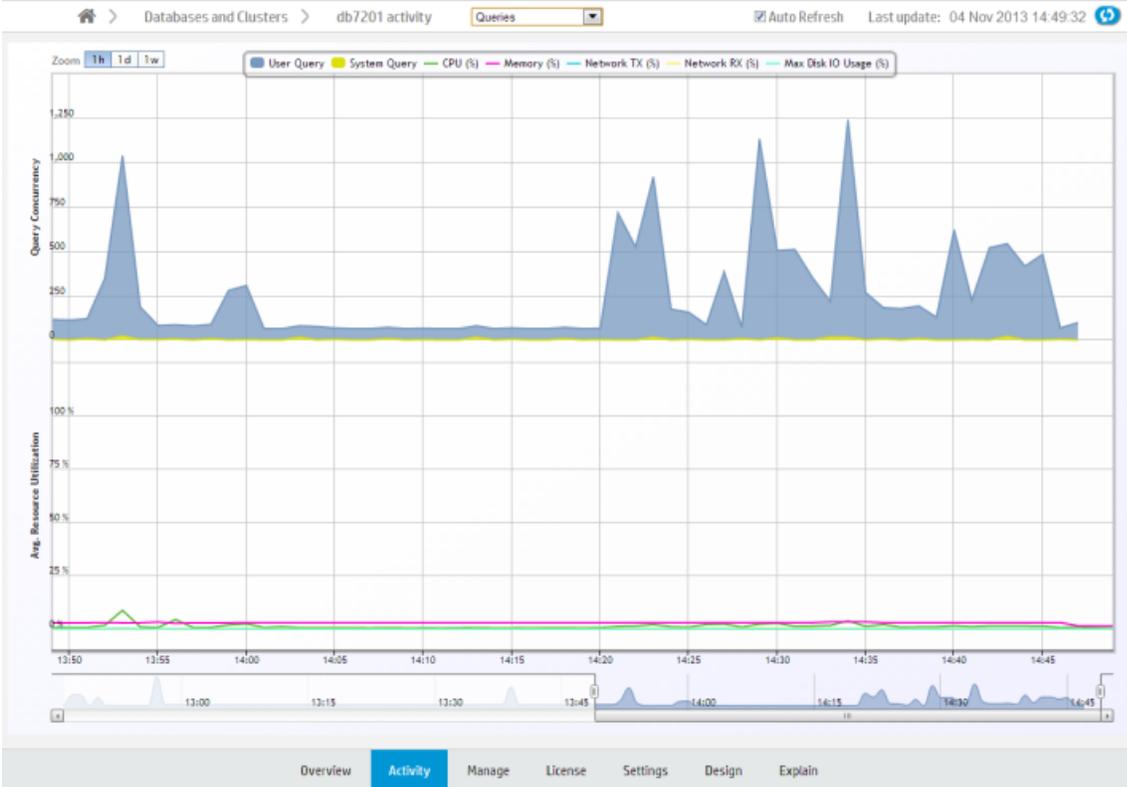


You can expand some charts to view them in larger windows. Click the expand icon in the title bar to do so:

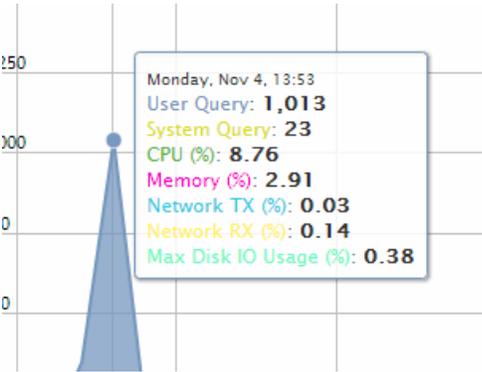


Monitoring Query Activity

The Queries chart displays information about query concurrency and average resource usage for CPU/memory, network activity, and disk I/O percent based on maximum rated bandwidth.



Hover over a data point for more information about percent usage for each of the resource types.



If you click a data point, MC opens a details page for that point in time, summarizing the number of user queries and system queries. This page can help you identify long-running queries, along with the query type. You can sort table columns and export the report to a file.

Monitoring Key Events

On the main Queries page, MC reports when a key event occurred, such as a Workload Analyzer or rebalance operation, by posting a **WLA** (Workload Analyzer) and/or **RBL** (rebalance) label on the resource section of the chart.

Filtering Chart Results

The default query concurrency is over the last hour. The chart automatically refreshes every five minutes, unless you clear the Auto Refresh option in the toolbar. You can filter results for 1 hour, 1 day, or up to 1 week, along with corresponding average resource usage. You can also click different resources in the legend to show or hide those resources.

To return to the main Queries page, use the slider bar or click the 1h button.

Viewing More Detail

To zoom in for detail, click-drag the mouse around a section or use the sliding selector bar at the bottom of the chart. After the detailed area displays, hover your cursor over a data point to view the resources anchored to that point in time.

For more detail about user or system queries, click a data point on one of the peaks. A **Detail** page opens to provide information about the queries in tabular format, including the query type, session ID, node name, query type, date, time, and the actual query that ran.

The bottom of the page indicates the number of queries it is showing on the current page, with Previous and Next buttons to navigate through the pages. You can sort the columns and export contents of the table to a file.

To return to the main Queries page, click **<database name> Activity** in the navigation bar.

Monitoring Internal Sessions

The Internal Sessions chart provides information about Vertica system activities, such as Tuple Mover and rebalance cluster operations, along with their corresponding resources, such as CPU/memory, networking, and disk I/O percent used.

Hover your cursor over a bar for more information. A dialog box appears and provides details.

Filtering Chart Results

You can filter what the chart displays by selecting options for the following components. As you filter, the **Records Requested** number changes:

- **Category**—Filter which internal session types (moveout, mergeout, rebalance cluster) appear in the graph. The number in parentheses indicates how many sessions are running on that operation.
- **Session duration**—Lists time, in milliseconds, for all sessions that appear on the graph. The minimum/maximum values on which you can filter (0 ms to *n* ms) represent the minimum/maximum elapsed times within all sessions currently displayed on the graph. After you choose a value, the chart refreshes to show only the internal sessions that were greater than or equal to the value you select.
- **Records requested**—Represents the total combined sessions for the Category and Session Duration filters.

Monitoring User Sessions

The User Sessions charts provide information about Vertica user activities for all user connections open to MC.

Choose **User Sessions** from the menu at the top of your database's Activity page to view these charts.

View Open Sessions

The Open Sessions tab displays a table of currently open sessions for each user. You can close a session or cancel a query on this tab by selecting that option from the **Actions** column.

Open Sessions | All Sessions | Sort Users | Toggle Columns

16 Total Open Sessions | 200 Max Client Sessions

Ben | Open Sessions: 1 | Max Sessions: unlimited | Idle Session Timeout: unlimited

Session Start	Session Duration	Current Request	Request Duration	Actions
Oct 26, 2016 2:46:18 PM	508s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	507s	-select-

Dasheng | Open Sessions: 1 | Max Sessions: unlimited | Idle Session Timeout: unlimited

Session Start	Session Duration	Current Request	Request Duration	Actions
Oct 26, 2016 2:45:35 PM	551s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	550s	-select-

Joe | Open Sessions: 4 | Max Sessions: unlimited | Idle Session Timeout: unlimited

Session Start	Session Duration	Current Request	Request Duration	Actions
Oct 26, 2016 2:53:27 PM	78s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	78s	-select-
Oct 26, 2016 2:51:11 PM	214s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	214s	-select-
Oct 26, 2016 2:49:21 PM	324s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	324s	-select-
Oct 26, 2016 2:52:27 PM	138s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	138s	-select-

Mark | Open Sessions: 1 | Max Sessions: unlimited | Idle Session Timeout: unlimited

Overview | Activity | Manage | Design | Load | Query Plan | License | Settings

Click any row to open a **Session Details** dialog that shows more extensive information about that session.

Session Details

General

Session ID: v_mcdemodb_node0004-758805:0x148eb

User: Sherry

Node: v_mcdemodb_node0004

Session Start: Oct 26, 2016 3:28:13 PM Session Duration: 440s

Transaction And Statement Details

Transaction ID: 58546795157023372

Transaction Start: Oct 26, 2016 3:28:13 PM

Statement ID: 1

Statement Start: Oct 26, 2016 3:28:13 PM Statement Duration: 440s

Currently Running Query

```
select count(*) from online_sales.online_sales_fact t1 join  
online_sales.online_sales_fact t2 on t1.product_key = t2.product_key  
group by t1.call_center_key,t2.online_page_key;
```

Client Details

Client Hostname: [REDACTED] Client PID: 2784130

Client Label:

Client Type: vsql (07.00.0300)

Client OS: Linux 2.6.32-431.20.3.el6.x86_64 x86_64

Client OS User Name:

Security Details

Close

To configure the Open Sessions page display:

- Use the **Sort Users** button at the top right of the page to sort by user name or number of open sessions.

- Use the **Toggle Columns** button at the top right of the page to select which columns to display. Each table displays session information by column, such as the session start time or the

Navigation: Databases and Clusters > MCDemoDB > Activity > User Sessions

Auto Refresh: [checked] Last update: [refresh icon]

Buttons: Sort Users, Toggle Columns

Summary: 16 Total Open Sessions, 200 Max Client Sessions

Session Start	Session Duration	Current Request	Request Duration	Actions
Ben Open Sessions: 1 Max Sessions: unlimited Idle Session Timeout: unlimited				
Oct 26, 2016 2:46:18 PM	508s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	507s	-select-
Dasheng Open Sessions: 1 Max Sessions: unlimited Idle Session Timeout: unlimited				
Oct 26, 2016 2:45:35 PM	551s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	550s	-select-
Joe Open Sessions: 4 Max Sessions: unlimited Idle Session Timeout: unlimited				
Oct 26, 2016 2:53:27 PM	78s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	78s	-select-
Oct 26, 2016 2:51:11 PM	214s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	214s	-select-
Oct 26, 2016 2:49:21 PM	324s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	324s	-select-
Oct 26, 2016 2:52:27 PM	138s	select count(*) from online_sales.online_sales_fact t1 join online_sales.online_sales_fact t2 on t1.product_key = t2.product_key ...	138s	-select-
Mark Open Sessions: 1 Max Sessions: unlimited Idle Session Timeout: unlimited				

Navigation: Overview | Activity | Manage | Design | Load | Query Plan | License | Settings

View All User Sessions

The All Sessions tab displays a history of all user sessions in a swim lane chart.



What Chart Colors Mean

Bars outlined with a dotted line are currently running sessions.

Sessions are divided into two colors, yellow and blue.

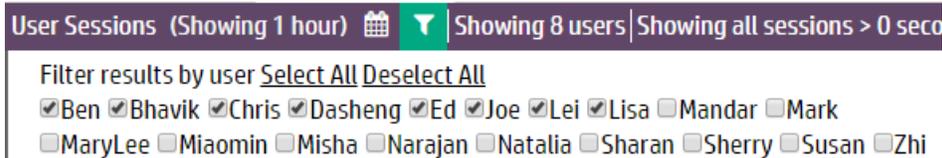
- Yellow bars represent user (system) sessions. If you click a yellow bar, MC opens a Detail page that shows all queries that ran or are still running within that session.
- Blue bars represent user requests (transactions within the session). If you click a blue bar in the graph, MC opens a Detail page that includes information for that query request only.

When you hover your mouse over a transaction bar, a dialog box provides summary information about that request, such as which user ran the query, how long the transaction took to complete, or whether the transaction is still running.

Filter Chart Results

Extremely busy systems will show a lot of activity on the interface, perhaps more than you can interpret at a glance. You can filter chart results in several ways:

- **Zoom.** The context chart at the bottom of the page highlights in blue which section of the All Sessions chart you are viewing. Click and drag the blue box left or right to view earlier or later user sessions. Click and drag the edges of the blue box to zoom in or out.
- **Select fewer users.** Click the filter icon () at the top of the page. A menu of a menu of all available users appears below. Deselect users to exclude from the chart.



- **Change the session duration (how long a session took to run).** Click the Filter icon () at the top of the page. The **Filter sessions and queries by duration** field appears below. Enter the minimum session length (in seconds) to display on the chart and click **Update**.



- **Specify a time frame.** Click the Calendar icon () at the top of the page to display the From and To fields. Using the fields, select the time frame to display in the chart and click **Update**.

Monitoring System Memory Usage

The Memory Usage chart shows how system memory is used on individual nodes over time. Information the chart displays is stored based on Data Collector retention policies, which a superuser can configure. See [Configuring Data Retention Policies](#).

The first time you access the Memory Usage chart, MC displays the first node in the cluster. MC remembers the node you last viewed and displays that node when you access the Activity page again. To choose a different node, select one from the Nodes drop-down list at the bottom of the chart. The chart automatically refreshes every five minutes unless you disable the Auto Refresh option.

Tip: On busy systems, the Node list might cover part of the graph you want to see. You can move the list out of the way by dragging it to another area on the page.

Types of System Memory

The Memory Usage chart displays a stacking area for the following memory types:

- swap
- free
- fcache (file cache)
- buffer
- other (memory in use by all other processes running on the system besides the main Vertica process, such as the MC process or agents)
- Vertica
- rcache (Vertica ROS cache)
- catalog

When you hover over a data point, a dialog box displays percentage of memory used during that time period for the selected node.

Monitoring System Bottlenecks

The System Bottlenecks chart helps you quickly locate performance bottlenecks on a particular node. The first time you access the Activity page, MC displays the first node in the cluster. To choose a different node, select one from the Nodes drop-down list at the bottom of the chart.



The System Bottlenecks chart reports what MC identifies as the most problematic resource during a given time interval. You can use this chart as a starting point for investigation.

How MC Gathers System Bottleneck Data

Every 15 minutes, MC takes the maximum percent values from various system resources and plots a single line with a data point for the component that used the highest resources at that point in time. When a different component uses the highest resources, MC displays a new data point and changes the line color to make the change in resources obvious. Very busy databases can cause frequent changes in the top resources consumed, so you might notice heavy chart activity.

In the following example, at 08:24 the maximum resources used changed from Disk I/O to CPU. The System Bottlenecks charts denotes this with a change in line color from brown to green.



The Components MC Reports on

MC reports maximum percent values for the following system components:

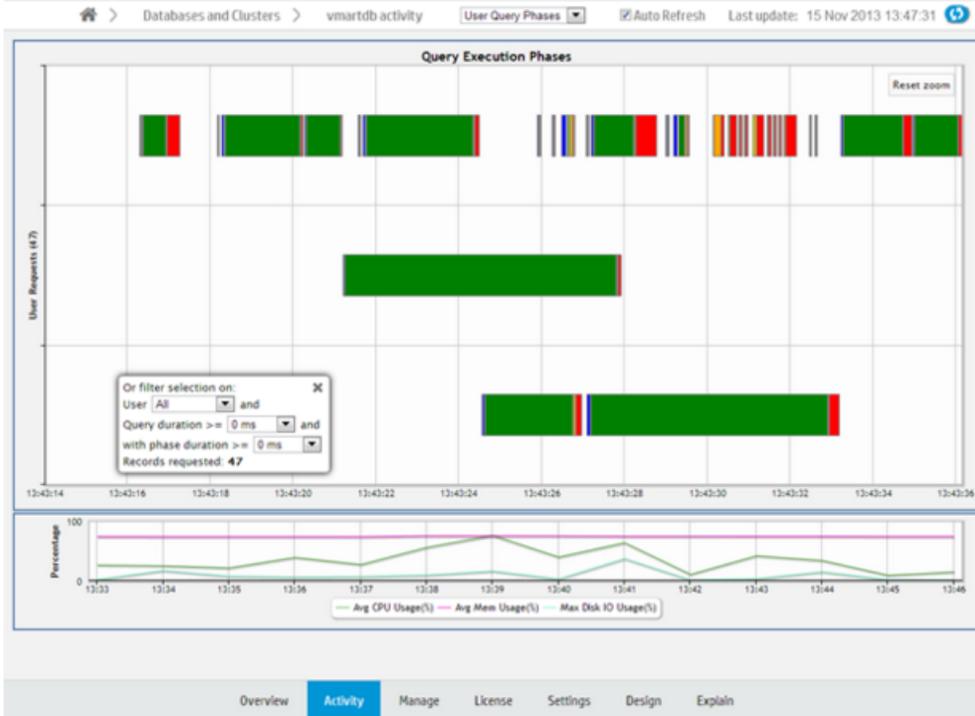
- Average percent CPU usage
- Average percent memory usage
- Maximum percent disk I/O usage
- Percent data sent over the network (TX)
- Percent data received over the network (RX)

How MC Handles Conflicts in Resources

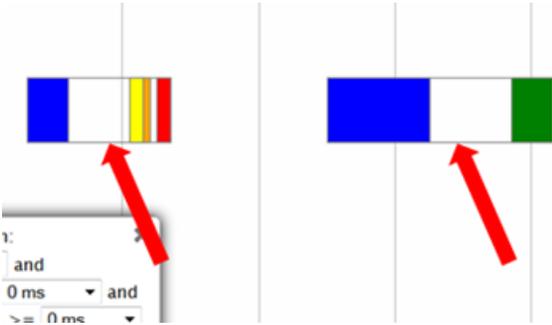
If MC encounters two metrics with the same maximum percent value, it displays one at random. If two metrics are very close in value, MC displays the higher of the two.

Monitoring User Query Phases

The User Query Phases chart provides information about the query execution phases that a query goes through before completion. Viewing this chart helps you identify at a glance queries possibly delayed because of resource contention.



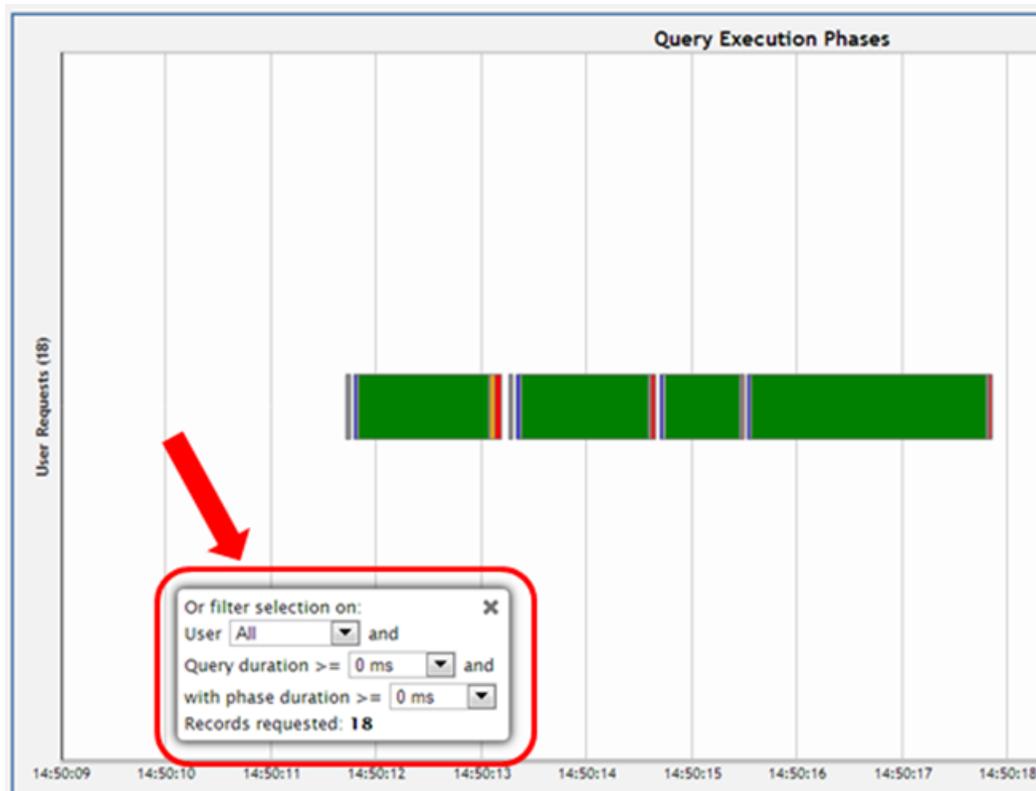
Each bar, bound by a gray box, represents an individual query. Within a query, a different color represents each query phase. The chart does not show phases for queries with durations of less than 4 seconds. Blank spaces within a query represent waiting times, as in the image below.



Hover over a phase in the query for information on the phase type and duration. The chart shows queries run over the last 15 minutes. The chart automatically refreshes every five minutes, unless you clear the Auto Refresh option in the toolbar.

Filtering Chart Results

You can filter what the chart displays by selecting options for the user running the query, minimum query duration, and minimum phase duration.



Viewing More Detail

To zoom in for detail, click-drag the mouse around a section of the chart. Click Reset Zoom, located at the top right corner of the chart, to restore the chart to its original view.

For more detail, click a query bar. The Detail page opens to provide information about the queries in tabular format, including the query type, session ID, node name, query type, date, time, the actual query that ran, and an option to run Explain Plan or profile the query. Click a table column header to sort the queries by that category.

To export the contents of the table to a file, click Export, located at the upper right of the page.

To return to the main Queries page, click Activity in the navigation bar.

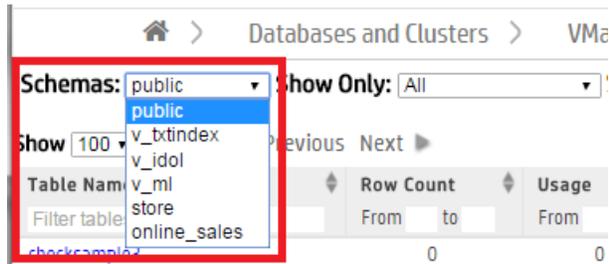
Monitoring Table Utilization and Projections

The Table Utilization activity page helps you monitor tables and projections in your database by schema.

You can use the table utilization charts on this page to identify outliers among your tables, such as those that are large or overused. The projections summary can help identify if projections are evenly distributed across nodes.

Table Visualizations

To specify a schema to view, choose one from the Schemas menu at the top of the activity page. The summary of tables and projections in that schema appear on the page.



MC visualizes your available tables by schema in a table chart or as a treemap chart.

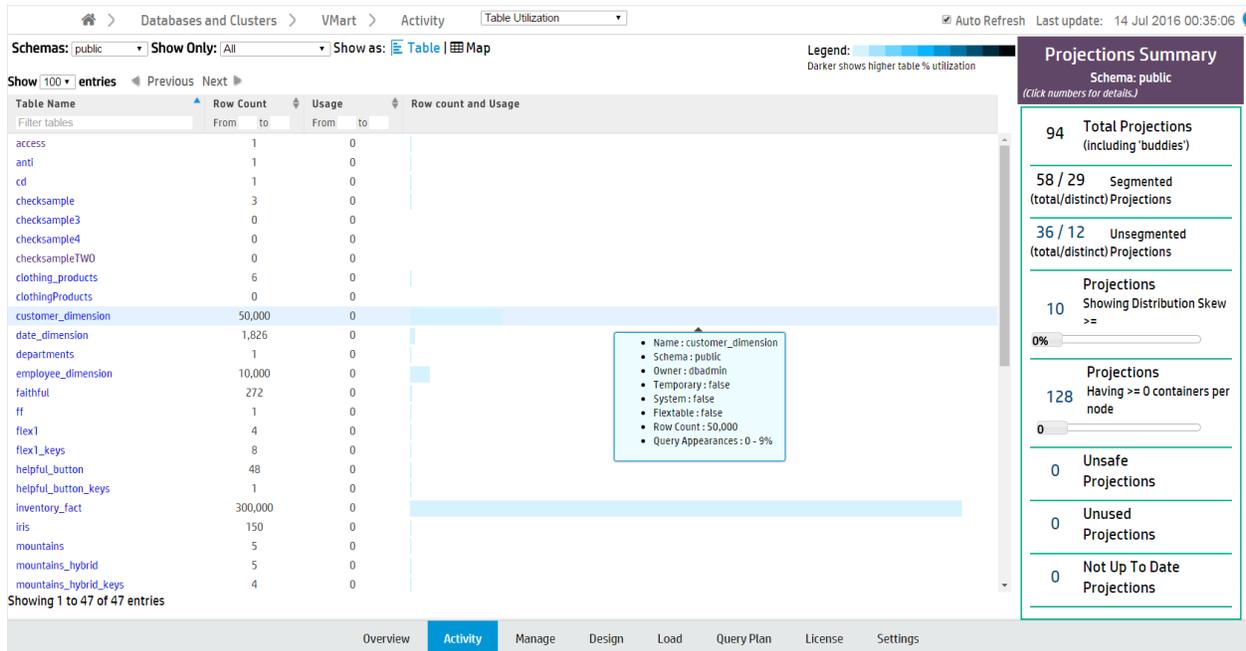
The Table chart provides flexible sorting to help you find tables you are interested in, especially if you have a large number of tables. You can use the Treemap chart for an at-a-glance graphical visualization of the relative size of your tables.

To choose how to visualize your table data, click **Table** or **Map** from the Show As menu. By default, MC displays the Table visualization. The current visualization is highlighted in blue in the Show As menu.

Show as:  **Table** |  Map

The Table chart, shown below, displays each table's row count, usage (percentage of time the table is queried), and a bar representing both row count and usage. The length of the bar indicates row count; a darker color indicates higher usage.

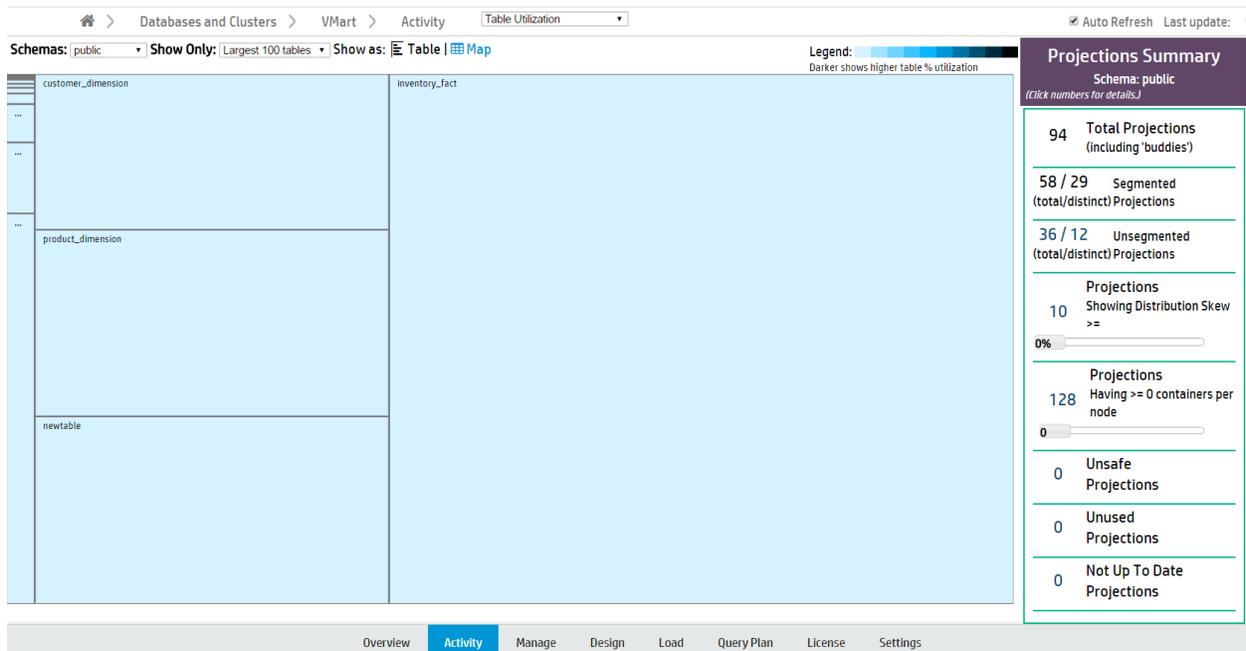
Hover over a bar in the chart to view details about that table, or click the table name to view its Table Details page.



In the Treemap visualization, tables are represented by boxes nested by size and colored by usage. Darker colors indicate higher table usage.

Depending on the number of tables in your database, the chart may be crowded. Use the Show Only filter at the top of the page to display only the largest or smallest 100 tables.

Hover over a table to view more details, or click to view its Table Details page.



The Table Details page displays information on the table's properties and columns, and about projections for the table. To the right, bar charts provide information, by node, on the amount of storage, number of deleted rows, and number of deleted vectors.

The screenshot displays the 'Table Properties For Public.Customer_dimension' section with the following data:

Property	Value
avg_bytes_per_row	43.38 bytes
is_flextable	
is_system_table	
is_temp_table	
owner_name	dbadmin
row_count	50000
table_name	customer_dimension

The 'Projections' section shows a table with the following columns:

Column Name	Data Type	Minimum	Maximum	Cardinality
annual_income	int	N/A	N/A	-1
customer_address	varchar(256)	N/A	N/A	-1
customer_age	int	N/A	N/A	-1
customer_city	varchar(64)	N/A	N/A	-1
customer_gender	varchar(8)	N/A	N/A	-1
customer_key	int	N/A	N/A	-1
customer_name	varchar(256)	N/A	N/A	-1
customer_region	varchar(64)	N/A	N/A	-1
customer_since	date	N/A	N/A	-1
customer_state	char(2)	N/A	N/A	-1
customer_type	varchar(16)	N/A	N/A	-1
deal_size	int	N/A	N/A	-1
deal_stage	varchar(32)	N/A	N/A	-1

The 'Storage By Node' chart shows three bars representing storage usage per node. The '# Deleted Rows By Node' and '# Delete Vectors By Node' charts show zero values for deleted rows and vectors.

Note: If you have deleted table rows recently, Management Console may not display the most recent row count. MC updates the row count when mergeout occurs. See [Mergeout](#).

Projections Summary

The projections summary displays information about projections by schema:

- Total projections.
- Segmented projections, the number of projections segmented across multiple nodes.
- Unsegmented projections, the number of projections that are not segmented across multiple nodes.
- Projections Showing Distribution Skew, the number of projections unevenly distributed across nodes. Tables with fewer than 1000 rows are not counted. Move the slider to configure filter by distribution skew percentage.
- The number of projections with more than the specified number of containers per node. Move the slider to configure the minimum number of containers.

- Unsafe projections, the number of projections with a K-safety less than the database's K-safety.
- Unused projections.

Click a projections number to view a list of the specified projections and their details.

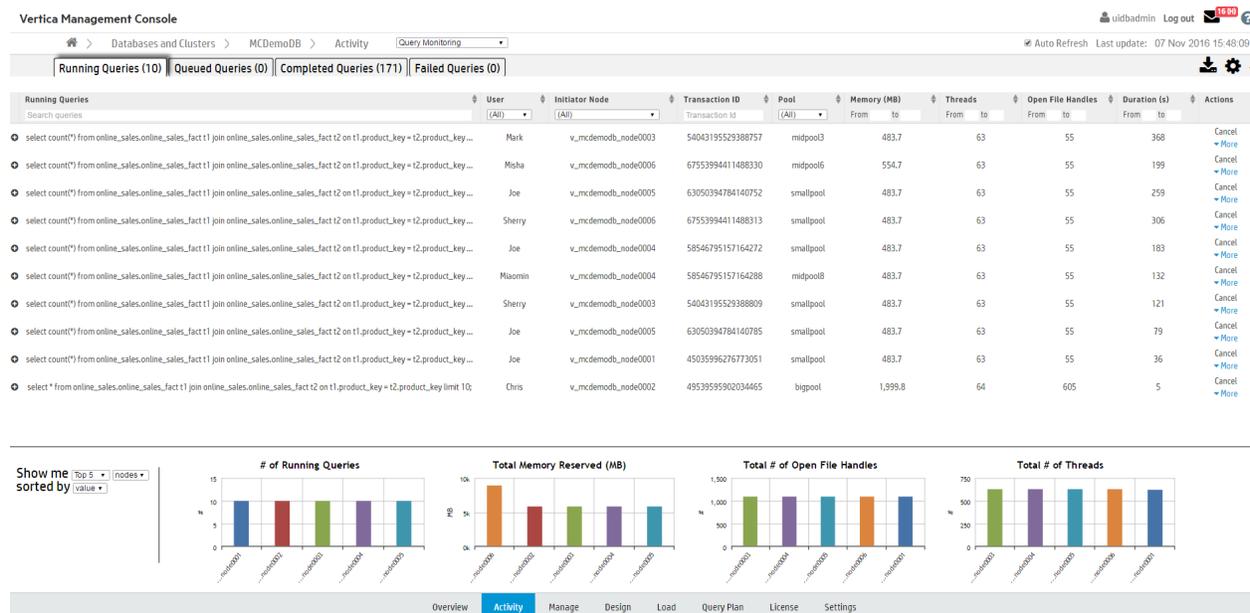
Monitoring Running Queries

The Query Monitoring activity page displays the status of recent and currently running queries, and resource information by node and user. From this page, you can profile a query or cancel a running query.

Use this page to check the status of your queries, and to quickly cancel running or queued queries to free up system resources. This page can help you identify where resources are being used, and which queries, users, or nodes are using the most resources.

The Query Monitoring page includes four tables, displayed as tabs:

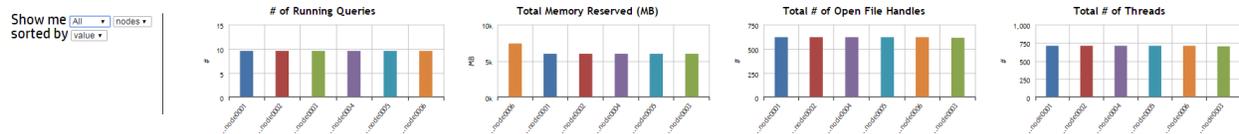
- Running queries
- Queued queries
- Completed queries
- Failed queries



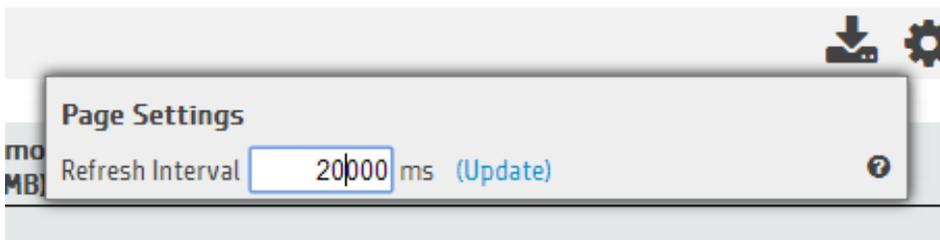
From the **Actions** column you can:

- **Cancel.** Cancel a running or queued query.
- **Close session.** Close a session for a running or queued query.
- **Explain.** Open the Query Plan page for any query.
- **Profile.** Profile any query on the Query Plan page.

The four bar charts at the bottom of the page display aggregate query usage by node or user. Hover over a bar with your cursor to see its value. When sorted by value, the left-most bar in each chart represents the node or user with the highest value.



The Query Monitoring page refreshes every 20 seconds by default. To change the refresh interval, click the Page Settings button in the upper-right corner of the page. A dialog appears. Type the new refresh interval, in milliseconds, in the text box.

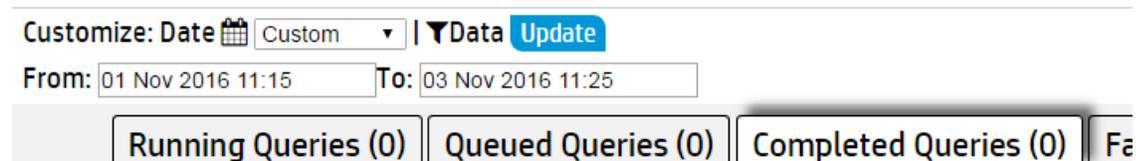


Filtering Chart Results

Use the search field below each column title to narrow down your chart results. (For example, if you enter the text *SELECT product_description* in the **Search Queries** field and select a specific node in the **Initiator Node** column, the chart returns only queries which both contain that text and were initiated on the node you specified.)

Click a column title to sort the order of the queries by that category.

There may be a large number of results for Completed and Failed Queries. Use the **Customize** section at the top of these two tabs to further filter your chart results. For either tab, you can select a custom date and time range for your results.

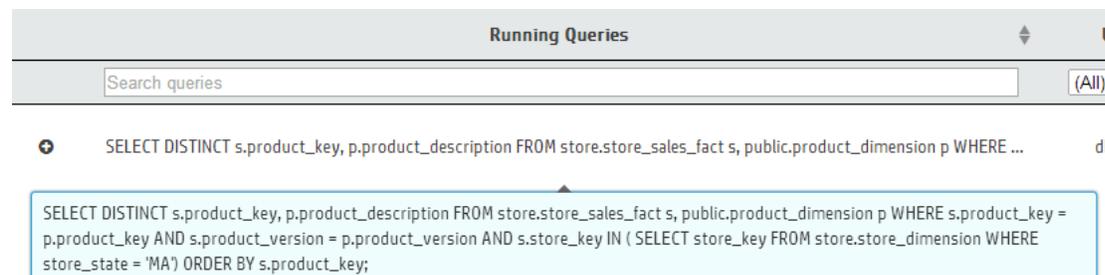


In the Completed Queries tab, click **Data** to enter additional query information to filter based on any of the following fields:

- User
- Request
- Request Duration
- Node
- Request label

Viewing More Details

Click a query to view the entire query.



In the Failed Queries chart, click the **plus (+)** icon next to a failed query to see the failure details for each node involved in the query's execution.

To export the data in one of the Query Monitoring tables, click the table's tab, then click the **Export** () button in the upper-right corner of the page. The browser downloads the data for that chart in a .dat file. The data exported includes columns that may not be visible in the MC, including the minimum and maximum values for Memory, Thread Count, and Open File Handles.

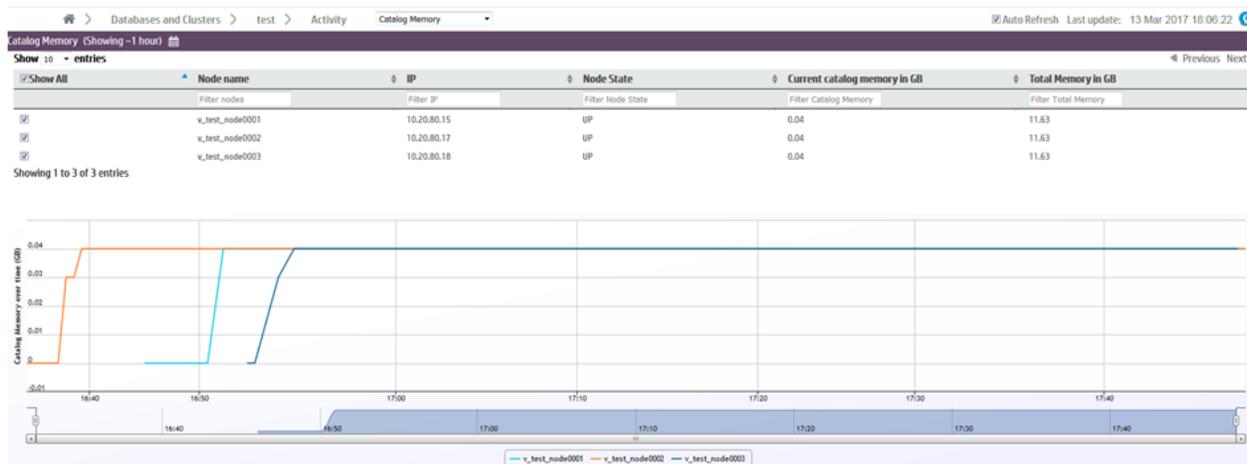
Monitoring Catalog Memory

The Catalog Memory activity page displays the catalog memory for each node. Use this page to check for sudden changes in catalog memory, or discrepancies in memory distribution across your nodes.

The Catalog Memory page displays:

- **A node details table.** The table lists the details of each node in the database, including their current catalog memory and total memory usage.
- **A catalog memory chart.** A line graph visualization of each node's catalog memory usage over time. Each line represents a node. The color legend at the bottom of the chart indicates the color of each node's line.

In the image below, catalog memory begins at 0GB for all three nodes. Over the next twenty minutes, catalog memory increases to 0.04GB in the second node (orange), then the first node (cyan), and finally in the third node (dark blue). Starting at 16:55, note that the three overlapping node lines appear as one line when all three nodes have the same catalog memory.



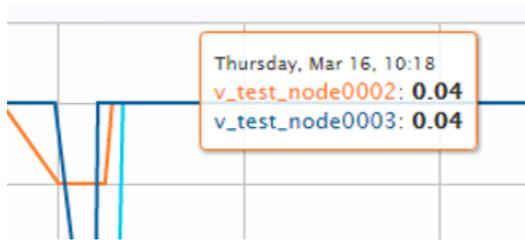
Filtering Chart Results

If you have many nodes in your database, you may want to display only certain nodes in the catalog memory chart. You can remove nodes from the chart in two ways:

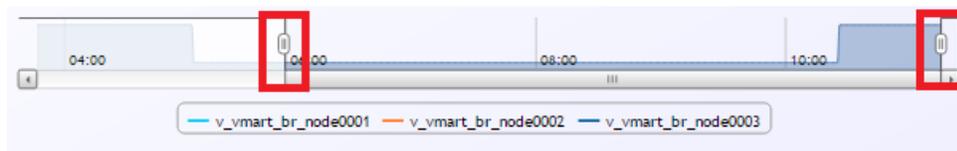
- Deselect the node's check box in the node details table.
- Deselect the node in the color legend below the chart.

Viewing More Details

Hover over any line in the chart to view the time, node name, and catalog size.



At the bottom of the chart is a summary bar that shows a quick overview of the catalog memory over time. Move the sliders on either side of the chart to zoom in on a specific time frame in the chart. When zoomed in, you can use the scrollbar to move forward or backward in time.

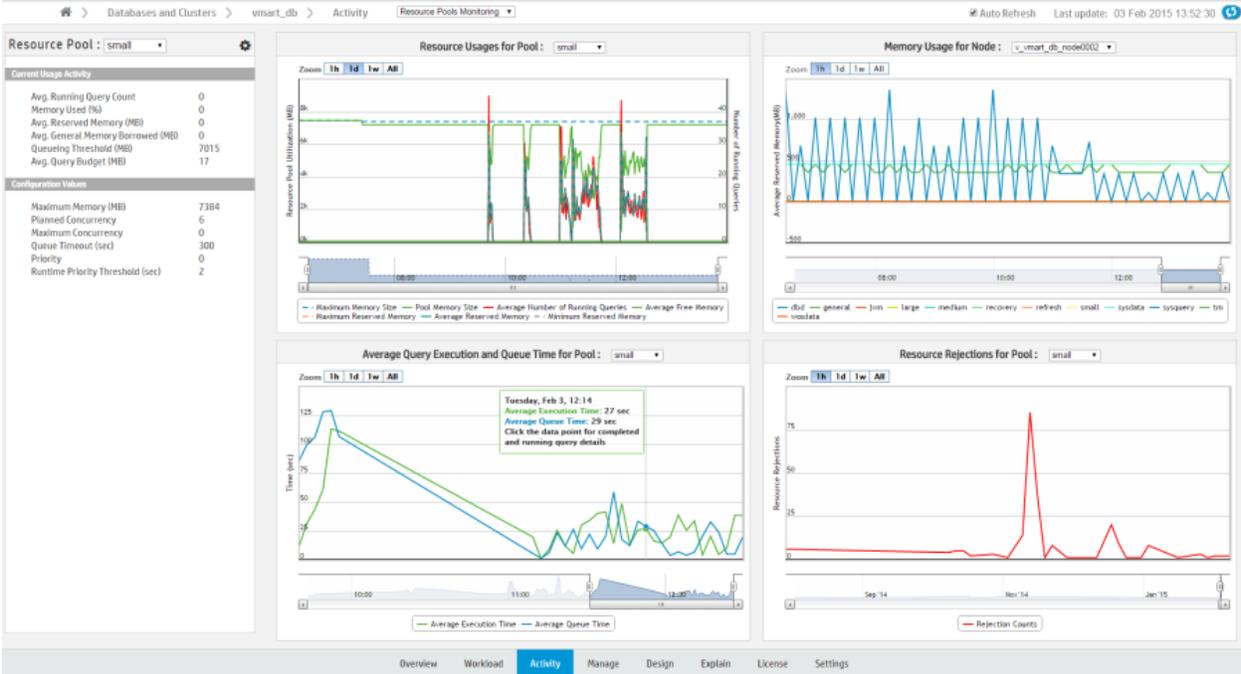


Monitoring Resource Pools

Management Console allows database administrators to monitor and configure resource pools through the Activity and Configuration pages. These pages help you manage workloads by providing visual representations of resource usage as well as resource pool configuration options.

Monitoring Resource Pools Charts

You can monitor your resource pools using the Resource Pools Monitoring charts, accessible through the Management Console Activity page.



Select a resource pool to view using the Resource Pool menu, located in the leftmost sidebar. In the sidebar, Current Usage Activity displays the pool's real-time statistics.

Monitor the selected resource pool using the following charts, which display the pool's historic data:

- **Resource Usages in Pool:** Shows the historically averaged acquired memory usage by each pool across all nodes. The graph uses two y-axes, one that shows memory size, and a second that shows the total number of running queries. Data is collected every hour. Hover over a data point for a summary of the memory usage at that specific point.
- **Memory Usage in Node:** Shows the historically acquired memory usages by all pools across all nodes. Data is collected every hour. Hover over a data point for a summary of the memory usage at that specific point
- **Average Query Execution and Query Time in Pool:** Shows the averaged query queue time plotted against the query execution time by each pool across all nodes. Data is collected every minute. Hover over data to get the average query execution and queue time in the specified pool. Click a data point to show detailed individual query information.
- **Resource Rejections in Pool:** Shows the historical total number of resource requests that were rejected in each pool across all nodes. Data is collected every hour. Click a data point to show rejection details and reasons in a pop-up window.

Configuring Resource Pools in MC

Database administrators can view information about resource pool parameters and make changes to existing parameters through the Management Console Configuration page. You can also create and remove new resource pools, assign resource pool users, and assign cascading pools.

See [Configuring Resource Pools in Management Console](#)

Permissions

Only the database administrator can monitor and configure resource pools in Management Console.

See Also

- [Configuring Resource Pools in Management Console](#)
- [Using Queries to Monitor Resource Pool Size and Usage](#)

Configuring Resource Pools in Management Console

Database administrators can view information about resource pool parameters and make changes to existing parameters in MC through the Resource Pools Configuration page. You can also create and remove new resource pools, assign resource pool users, and assign cascading pools.

Access the Resource Pools Configuration page from the Settings page by selecting the Resource Pools tab.

You can also access the Configuration page from the Resource Pools Monitoring chart, accessible through the Management Console Activity page. Click the tools icon on the top of the leftmost sidebar.

Permissions for Monitoring and Configuring Resource Pools

Only the database administrator can monitor and configure resource pools in Management Console.

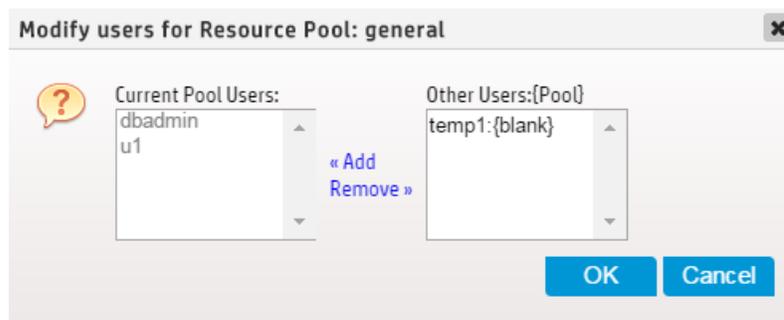
Modify Resource Pool Parameters

1. On the Resource Pools Configuration page, choose a resource pool from the Resource Pools field. Parameter fields for that resource pool appear.
2. Use the parameter fields to view and modify parameters for the resource pool. Hover your cursor in the parameter field to display information about that parameter.
3. Click **Apply** to save your changes. A success message appears

Modify Resource Pool Users

To add or remove resource pool users:

1. On the Resource Pools Configuration page, select a resource pool from the Resource Pools field.
2. Next to the Pool Users field, click **Add/Remove Pool Users**. The Modify Users for Resource Pool dialog appears.



3. The dialog displays users assigned to the resource pool in the Current Pool Users list. The Other Users list displays all other resource pool users are displayed, along with the pool to which they are currently assigned.

- a. To add users to the resource pool: Select the desired users from Other Users list and click **Add**.
 - b. To remove users from the resource pool: Select the users to be removed from the Current Pool Users list and click **Remove**.
4. Click **Apply** to save your changes. A success message appears.

Create and Remove Resource Pools

Database administrators can use MC to create resource pools and assign resource pool users, and remove user-generated resource pools.

To create a resource pool:

1. On the Resource Pools Configuration page, click **Create Pool**. Fields pre-populated with pool parameter default values appear.
2. Enter the new resource pool's parameters in the fields.
3. Click **Create Pool**. A success message appears.

To remove a resource pool:

1. First, remove all users from the resource pool to be deleted. This can be done on the Resource Pool Configuration Page.
2. When all users have been removed from the resource pool, choose the resource pool from the Resource Pools field on the Resource Pool Configuration Page. Parameter fields for that resource pool appear.
3. Click **Remove Pool**. A Confirm dialog appears.
4. Click **OK** in the Confirm dialog. A success message appears.

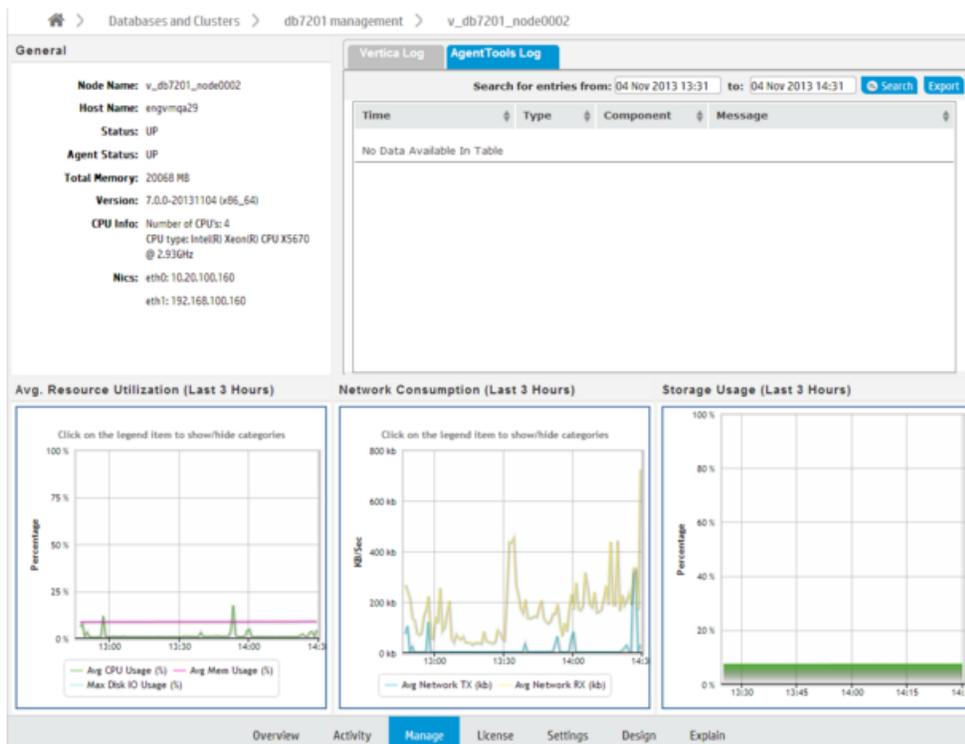
See Also

- [CREATE RESOURCE POOL](#)
- [Monitoring Resource Pools](#)
- [Using Queries to Monitor Resource Pool Size and Usage](#)

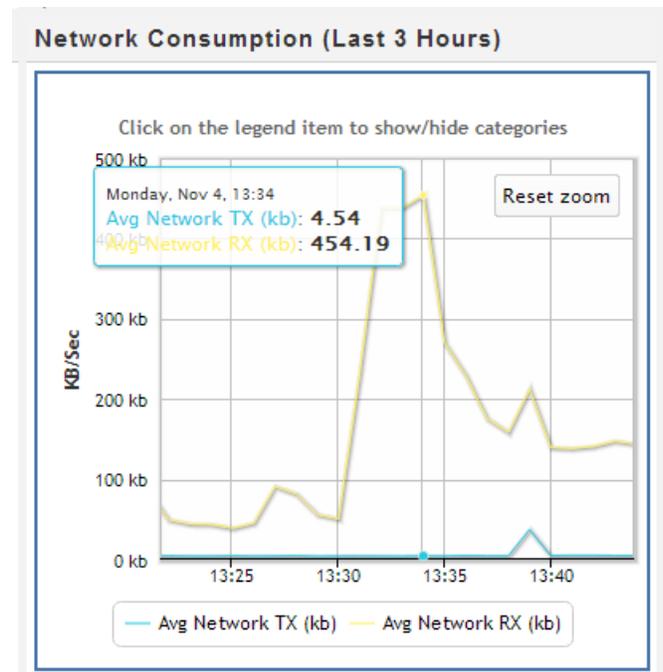
Monitoring Node Activity

If a node fails on an MC-managed cluster or you notice one node is using higher resources than other cluster nodes—which you might observe when monitoring the [Overview page](#)—open the [Manage](#) page and click the node you want to investigate.

The Node Details page opens and provides summary information for the node (state, name, total memory, and so on), as well as resources the selected node has been consuming for the last three hours, such as average CPU, memory, disk I/O percent usage, network consumption in kilobytes, and the percentage of disk storage the running queries have been using. You can also browse and export log-level data from AgentTools and Vertica log files. MC retains a maximum of 2000 log records.



For a more detailed view of node activity, use the mouse to drag-select around a problem area in one of the graphs, such as the large spike in network traffic in the above image. Then hover over the high data point for a summary.



See Also

- [Viewing the Overview Page](#)
- [Monitoring Cluster Performance](#)

Monitoring Database Messages in MC

As Management Console monitors your database, it periodically checks your system's health and performance. MC then generates messages to alert you about the state of your system. You can view and manage these messages in the Message Center.

Getting New Message Notifications

The Messages icon appears in the top-right of any database-specific page on MC (such as the Overview page or Activity page). The icon displays a new messages badge. The badge displays how many highest priority messages you have received recently.

The messages badge displays the color and letter of that priority:

- Red (H): High Priority
- Orange (N): Needs Attention
- Blue (I): Informational

In addition, the Thresholds Notification widget on the Overview page displays a summary of alerts about exceeded thresholds. Only high-priority threshold alerts appear in the Thresholds Notification widget. You can set the priority of alerts for different thresholds in the Thresholds tab on your database's Settings page.

Viewing Message Center

You can look at a preview of your most recent messages without navigating away from your current page. Click the Message Center icon in the top-right corner of Management Console to view the notification menu. The menu appears as a drop-down preview of your inbox. From this menu, you can delete, archive, or mark your messages as read.

dbadmin Log out IDOL  

[Message Center >](#)
✓ Archive All

Highest Priority Messages: (Showing latest 1 of 1)

Critical		Feb 29 09:16:11	Failed to create webhook for host 10.20.100.247
-----------------	--	-----------------	---

Lower Priority Messages: (Showing latest 9 of 9) ✓ Archive All

Notice	VMart	Mar 03 05:09:06	Analyze Workload operation started on Database	✓ ✕
Notice	VMart	Mar 02 05:09:39	Analyze Workload operation has succeeded on Database	
Notice	VMart	Mar 02 05:09:32	Analyze Workload operation started on Database	
Notice	VMart	Mar 01 05:03:28	Analyze Workload operation started on Database	
Notice	VMart	Feb 29 09:17:18	Analyze Workload operation has succeeded on Database	
Notice	VMart	Feb 29 09:17:12	Analyze Workload operation started on Database	
Info	VMart	Feb 29 09:17:11	Agent status is UP on IP 10.20.100.249	
Info	VMart	Feb 29 09:17:11	Agent status is UP on IP 10.20.100.248	
Info	VMart	Feb 29 09:17:11	Agent status is UP on IP 10.20.100.247	

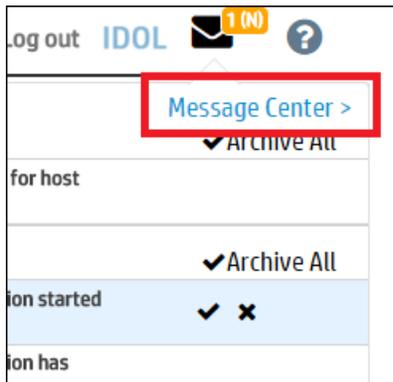
You can access the Message Center in three ways:

- Click the **Message Center** button on the MC Home Page. The Message Center displays messages about the most recent database you have viewed.

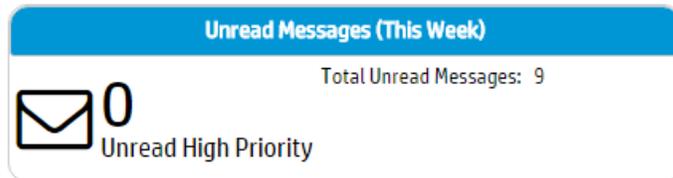
Manage Information

 Existing Infrastructure Manage databases and clusters	 Message Center Go to the database message center
 Diagnostics View diagnostics and support information	 MC Settings Manage application and user settings

- Click the **Messages** icon in the top-right of any database-specific page on MC. (For example, the icon appears on the Overview and Activity pages). A menu of your recent messages appears. Click **Message Center** in the top-right of the menu to reach the Message Center for the database you most recently viewed.



- Click the **Messages** widget on the right side of the Overview page.



Filtering Recent Messages

By default, Message Center displays up to 600 of a database's most recent messages from the past week.

Message Center

Select All Select None Mark Read Delete Msgs Export Alerts

Showing: (All DBs) 9 2226 2250
High Priority Need Attention Informational

Retrieve Older Messages

Recent Messages Date: Today 393 Archive All Delete all

Warning	chris360B	Threshold : Node CPU < 10 %	09 Nov 2015 16:01:19
Critical	chris360B	Threshold : Node Memory > 10 %	09 Nov 2015 16:01:19
Critical	chris360B	Threshold : Node Disk I/O < 10 %	09 Nov 2015 16:01:19
Warning	chris360B	Threshold : Node Disk Usage > 10 %	09 Nov 2015 16:01:18
Critical	chris360B	Threshold : Node Memory > 10 %	09 Nov 2015 15:51:19
Warning	chris360B	Threshold : Node CPU < 10 %	09 Nov 2015 15:51:19
Critical	chris360B	Threshold : Node Disk I/O < 10 %	09 Nov 2015 15:51:19
Warning	chris360B	Threshold : Node Disk Usage > 10 %	09 Nov 2015 15:51:18
Warning	chris360B	Threshold : Node CPU < 10 %	09 Nov 2015 15:41:19
Critical	chris360B	Threshold : Node Memory > 10 %	09 Nov 2015 15:41:19
Critical	chris360B	Threshold : Node Disk I/O < 10 %	09 Nov 2015 15:41:19
Warning	chris360B	Threshold : Node Disk Usage > 10 %	09 Nov 2015 15:41:18

To see more messages:

1. Click **Retrieve Older Messages** in the top-right of the page.
2. Use the drop-down **From** and **To** calendars to select a time range from which to retrieve older messages. You can specify days, hours, and minutes.
3. Click **Search** to display the messages from the specified time frame.

Depending on your database, you may have received hundreds of recent messages in the past seven days. To help you find the messages most relevant to you, Message Center provides several ways to filter recent messages:

- **By search term:** Use the search field at the top of the page to filter messages by any search term, such as description or date.
- **By database:** View a specific database's messages by choosing it from the Showing: [database] drop-down list at the top of the page. Message Center's default view lists up to 600 of the most recent messages for each database MC monitors.
- **By time period:** Message Center groups your recent messages by Today, Yesterday, and This Week. Click any of these headings to collapse that section of messages.
- **By message priority:** Messages are categorized as High Priority, Needs Attention, or Informational. The number of messages of each priority level appears at the top-right of the page. By default, all priority levels are displayed in the inbox. To filter messages of one priority level out of your current view, click a message count value at the top of the inbox to deselect it.
- **By threshold messages:** MC generates messages indicating when certain thresholds are exceeded in your database. To view only messages related to exceeded thresholds, click the

Thresholds Messages tab on the left hand side of the page. To modify these alert thresholds, go to the Thresholds tab, which appears in your database's Settings page.

By default, Message Center displays up to 600 recent entries per database. You can adjust the maximum limit of messages the Message Center displays in the `/opt/vconsole/config/console.properties` file. To adjust the limit, change the number in the following line:

```
messageCenter.maxEntries=600
```

Managing and Archiving Messages

Message Center archives all messages you mark as read. To mark messages as read or unread, select a message or multiple messages in the Message Center and click **Mark Read** or **Mark Unread**.

You can also perform the following tasks:

- See an archived message—Click **Archived Messages** on the right side of the Message Center. Then, use the **From** and **To** fields at the top of the page to select a date range from which to display archived messages.
- Sort archived messages by Type, Database Name, Description, or Date—Click any filter at the top of the Message Center.
- Delete messages permanently—If you prefer to delete messages, rather than archiving them, select the messages to delete. Then, click **Delete Msgs**.

Viewing Message Details

Within the Message Center, click the plus (+) symbol next to a message to get more information about the issue. You can also query the [V_MONITOR.EVENT_CONFIGURATIONS](#) table to get information about events. See the [SQL Reference Manual](#) for details.

Message Priority Levels

MC sorts messages by priority level and prioritizes them with color codes:

- High Priority (Red)
- Needs Attention (Orange)
- Informational (Blue)

At the top of your inbox, the Message Center displays the number of messages that fall within each level of priority.

Click any of the values to deselect them. This filters that type of message out of your current inbox view. For example, suppose you want to view the High Priority messages in your inbox. To remove lower-priority messages, you can click the message count values for the categories labeled Needs Attention and Informational. Doing so removes them from your current view.

Messages in the inbox are additionally labeled with one of seven subcategories, according to priority:

- High Priority: Emergency (0), Alert (1)
- Need Attention: Critical (2), Error (3)
- Informational: Warning (4), Info (5), Notice (6)

See Also

- [Customizing Threshold-Based Notifications](#)
- [Exporting MC-managed Database Messages and Logs](#)

Message Types

MC generates both default notifications and customizable threshold notifications.

Threshold Notifications

MC generates threshold-based notifications when the database exceeds specified limits on the following thresholds.

- License Status
 - License Usage
- System Health
 - Node Health
 - Node CPU
 - Node Memory
 - Node Disk Usage
 - Node Disk I/O
 - Node CPU I/O Wait
 - Node Reboot Rate
 - Node Catalog Memory
 - Node State Change
 - Network Health
 - Network I/O Error
- System Performance
 - Query
 - Queued Query Number
 - Failed Query Number
 - Spilled Query Number
 - Retried Query Number
 - Query Running Time
 - Resource Pool
 - Queries Reaching the Max Allowed Execution Time
 - Queries With Resource Rejections
 - Ended Query With Queue Time Exceeding Limit
 - Ended Query With Run Time Exceeding Limit

MC provides default settings for these notifications. You can customize threshold notification settings in the Thresholds tab in the database Settings page. Threshold notifications appear in the Message Center. High-priority threshold alerts also appear on the Overview Page. See [Customizing Threshold-Based Notifications](#) for how to prioritize alerts.

Default Notifications

By default, MC also generates messages about the database that appear only in the Message Center. You might receive messages about the following database-related conditions:

- Low disk space
- Read-only file system
- Loss of K-safety
- Current fault tolerance at critical level
- Too many ROS containers
- WOS overflow
- Change in node state
- Recovery error
- Recovery failure
- Recovery lock error
- Recovery projection retrieval error
- Refresh error
- Refresh lock error
- Workload Analyzer operations
- Tuple Mover error
- Timer service task error
- Last Good Epoch (LGE) lag

- License size compliance
- License term compliance

Customizing Threshold-Based Notifications

Management Console can generate notifications when your database exceeds threshold limits that you specify. You can configure threshold notifications per database in the Thresholds tab, which appears on your database's Settings page.

You can enable and modify message thresholds for each database that MC monitors. MC then generates a message indicating when a threshold is exceeded. For example, you can set the threshold for node disk usage to a 20% minimum and 80% maximum. As MC monitors node disk usage, it notifies you when any nodes exceed those minimum or maximum thresholds.

MC categorizes customizable message thresholds by license use, system health, and system performance. See [Message Types](#) for a list of the customizable thresholds available in MC.

The default threshold notification settings for Resource Pool Monitoring apply to the GENERAL pool. To customize messages for custom pools, click **Add New Threshold for Custom Resource Pools**.

Configuring Settings for Thresholds

Message thresholds have the following configurable settings:

- Enabling and disabling threshold-specific messages
- Threshold values
- Check time interval
- Alert Priority
- Email Destination
- Email Interval

Configure your notification settings, and click **Apply** to save your changes. If a message threshold has not been set previously, MC displays the default threshold for that setting.

Setting Priorities

You can set a notification to one of three priorities. If a threshold value is exceeded, MC:

- Priority 1 — Sends you an email notification, displays a message in the Overview page, and creates a message in the Message Center.
- Priority 2 — Displays a message in the Overview page, and creates a message in the Message Center.
- Priority 3 — Creates a message in the Message Center.

Setting Email Notifications

In order for MC to send email notifications, you must first provide MC with SMTP server settings. See [Set Up Email](#) .

When you set a threshold to Priority 1, MC can send email notifications to subscribed users.

To subscribe users to a threshold email message:

1. Set the threshold's Alert Priority field to **Priority 1: Overview and Email**. The Email Destination and Email Interval fields appear.
2. Click the browsing icon next to the Email Destination field. The Subscriber List dialog box opens.
3. You can add emails in one of two ways:
 - Select from the list of existing MC users with associated email addresses. (See [Managing MC Users](#) for how to add email addresses to user profiles.)
 - Provide an additional email address in the Entering New Email field. Click the plus (+) icon next to the field to add the address.
4. Click **OK**.
5. Select an option from the **Email Interval** field to choose how frequently users receive email notifications about the threshold if it is exceeded.
6. Click **Apply** at the top-right of the page to save your email subscriber settings.

See Also

- [Message Types](#)
- [Set Up Email](#)
- [Managing MC Users](#)

Set Up Email

Management Console can generate email alerts about high-priority database thresholds. To receive email alerts, you must first configure your SMTP settings in MC.

You must be an administrator to provide SMTP settings.

To set up MC to send email:

1. Select the **Email Gateway** tab on the MC Settings page.
2. Provide the following information about your SMTP server:
 - SMTP Server (Hostname). Maximum length is 255 characters. You can enter either the hostname or IP address.
 - SMTP server port.
 - Session Type (SSL or TLS).
 - SMTP Username.
 - SMTP Password.
 - Originating Email Alias. MC sends alerts from the email address you provide.
3. Click **Test** at the top of the page. MC validates your SMTP settings and sends a test email to the inbox of the email alias you provided.
4. Verify that you successfully received the test email.
5. Click **Apply** at the top-right of the page to save the settings.

With email settings completed, you can configure MC to send high-priority threshold alerts through email. See [Customizing Threshold-Based Notifications](#).

Searching Database Messages Managed by MC

The Management Console Message Center displays the 10,000 most recent database messages, starting with the most recent record.

If more than 600 recent messages exist for a database, MC returns a message letting you know that only the most recent 600 entries are shown, so you have the option to filter.

Changing Message Search Criteria

You can use the search field at the top right of the Message Center to filter your messages by any search term, such as database name, description, or date.

Additionally, click on any filter at the top of the Message Center to sort message by Type, Database Name, Description, or Date. To further filter messages, you can select a database from the Database Name drop down list, type a keyword in the Description filter, or enter a range of dates in the Date filter.

Retrieving Additional Messages

If more than 10,000 messages exist, older messages that occurred before the 10,000 messages do not appear in the Message Center by default. You can click **Retrieve Additional Alerts** to filter for these older messages.

To specify which messages to retrieve, click **Retrieve Additional Alerts**. The Search for Messages dialog appears. Choose a date range, one or more message types, or one or more message types within a specific date range. Click **OK** to view the messages in that range.



Search for messages [X]

Search by date range:
from: to:

Select message types to include:

Emergency Alert Critical
 Error Warning Notice
 Info

OK Cancel

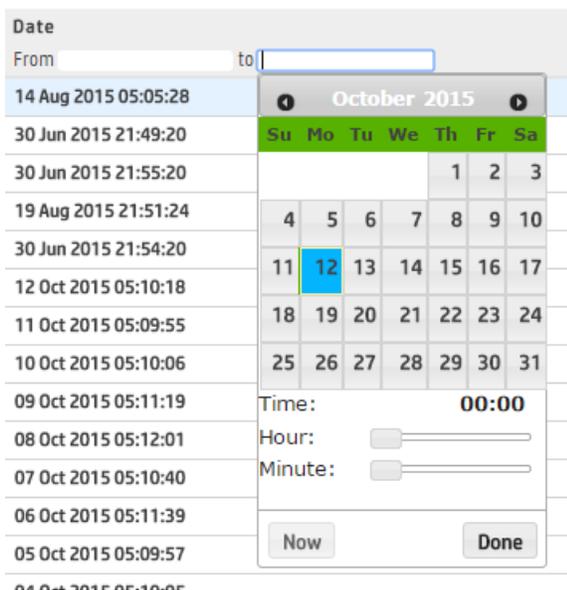
You can adjust the maximum limit of recent messages the Message Center displays in the `/opt/vconsole/config/console.properties` file. To adjust the limit, change the number in the following line:

```
messageCenter.maxEntries=5000
```

Specifying Date Range Searches

For date range searches, MC starts at the beginning of the time range and either returns all messages up to the specified end time or 10,000 messages, whichever comes first. You can filter message searches on the following date ranges:

- Any date-to-date period, including hour and minute
- Any time period up to now (forward searches)
- Any time period before now (backward searches)



After you specify a date range, click **Done** to close the calendar, and then click **OK** to see the results in the Message Center.

Exporting MC-managed Database Messages and Logs

You can export the contents of database messages, log details, query details, and MC user activity to a file.

Information comes directly from the MC interface. This means that if the last five minutes of `vertica.log` information displays on the interface, you can save that five minutes of data to a file, not the entire log. When you filter messages or logs, MC exports only the filtered results.

Depending on how you set your browser preferences, when you export messages you can view the output immediately or specify a location to save the file. System-generated file names include a timestamp for uniqueness.

The following table shows, by record type, the MC pages that contain content you can export, the name of the system-generated file, and what that file's output contains:

Message type	Where you can export on MC	System-generated filename	Contents of exported file
All db-related message types	Message Center page	<code>vertica-alerts-<timestamp>.csv</code>	Exports messages in the Message Center to a <code>.csv</code> file. Message contents are saved under the following headings: <ul style="list-style-type: none"> • Create time • Severity • Database • Summary (of message) • Description (more details)
MC log files	Diagnostics page	<code>mconsole-<timestamp>.log</code>	Exports MC log search results from MC to a <code>.log</code> file under the following headings: <ul style="list-style-type: none"> • Time • Type (message severity)

Message type	Where you can export on MC	System-generated filename	Contents of exported file
			<ul style="list-style-type: none"> • Component (such as TM, Txn, Recover, and so on) • Message
Vertica logs	<p>Manage page</p> <p>Double-click any node to get to the details and then click the VerticaLog tab</p>	vertica-vertica- <i><db></i> - <i><timestamp></i> .log	<p>Exports vertica log search results from MC to a .log file under the following headings:</p> <ul style="list-style-type: none"> • Time • Type (message severity) • Component (such as TM, Txn, Recover, and so on) • Message
Agent logs	<p>Manage page</p> <p>Click any node to get to the details and then click the AgentTools Log tab.</p>	vertica-agent- <i><db></i> - <i><timestamp></i> .log	<p>Exports agent log search results from MC to a .log file under the following headings:</p> <ul style="list-style-type: none"> • Time • Type (message severity) • Component (such as TM, Txn, Recover, and so on)

Message type	Where you can export on MC	System-generated filename	Contents of exported file
			<ul style="list-style-type: none"> Message
Query details	<p>Activity page</p> <p>Click any query spike in the Queries graph to get to the Detail page.</p>	vertica-querydetails-<db>-<timestamp>.dat	<p>Exports query details for the database between <timestamp> and <timestamp> as a tab-delimited .dat file. Content is saved under the following headings:</p> <ul style="list-style-type: none"> Query type Session ID Node name Started Elapsed User name Request/Query
MC user activity	<p>Diagnostics page</p> <p>Click the Audit Log task</p>	vertica_audit<timestamp>.csv	<p>Exports MC user-activity results to a .csv file. Content is saved under the following headings:</p> <ul style="list-style-type: none"> Time MC User Resource Target User Client IP

Message type	Where you can export on MC	System-generated filename	Contents of exported file
			<ul style="list-style-type: none">• Activity

Monitoring MC User Activity Using Audit Log

When an MC user makes changes on the MC interface, whether to an MC-managed database or to the MC itself, their action generates a log entry that records a timestamp, the MC user name, the database and client host (if applicable), and the operation the user performed. You monitor user activity on the **Diagnostics > Audit Log** page.

MC records the following types of user operations:

- User log-on/log-off activities
- Database creation
- Database connection through the console interface
- Start/stop a database
- Remove a database from the console view
- Drop a database
- Database rebalance across the cluster
- License activity views on a database, as well as new license uploads
- Workload Analyzer views on a database
- Database password changes
- Database settings changes (individual settings are tracked in the audit record)
- Syncing the database with the cluster (who clicked Sync on grid view)
- Query detail viewings of a database
- Closing sessions

- Node changes (add, start, stop, replace)
- User management (add, edit, enable, disable, delete)
- LDAP authentication (enable/disable)
- Management Console setting changes (individual settings are tracked in the audit record)
- SSL certificate uploads
- Message deletion and number deleted
- Console restart from the browser interface
- Factory reset from the browser interface

Background Cleanup of Audit Records

An internal MC job starts every day and, if required, clears audit records that exceed a specified timeframe and size. The default is 90 days and 2K in log size. MC clears whichever limit is first reached.

You can adjust the time and size limits by editing the following lines in the `/opt/vconsole/config/console.properties` file:

```
vertica.audit.maxDays=90vertica.audit.maxRecords=2000
```

Filter and Export Results

You can manipulate the output of the audit log by sorting column headings, scrolling through the log, refining your search to a specific date/time and you can export audit contents to a file.

If you want to export the log, see [Exporting the User Audit Log](#).

If You Perform a Factory Reset

If you perform a factory reset on MC's Diagnostics page (restore it to its pre-configured state), MC prompts you to export audit records before the reset occurs.

Monitoring External Data Sources in Management Console

By default, Management Console monitors a database using information from that database's Data Collector (DC) tables. MC can also monitor DC tables you have copied into Vertica tables, locally or remotely.

MC administrators provide mappings to local schemas or to an external database containing the corresponding DC data. MC can then render its charts and graphs from the new repository instead of from local DC tables. This offers the benefit of loading larger sets of data faster in MC, and retaining historical data long term.

Note: MC also offers [External Monitoring](#), which allows you to set up a Vertica storage database through the MC interface, then use Kafka to stream your data to the storage database. You can use the Data Source mapping process below if you prefer to set up your own alternative data source, or do not plan to use Kafka streaming.

Map an Alternative Data Source

1. On the MC Settings page, navigate to the Data Source tab.
2. Select the database for which you are creating the data source mapping.
3. Choose the database user for which you want to create the mapping.
4. Set Repository Location to Local or Remote.
5. If Remote is selected, provide JDBC connection parameters for the remote database repository. Click **Validate Connection Properties** to confirm a successful connection.
6. Enter the schema mappings for v_internal and v_catalog. MC does not support mapping the v_monitor schema.
7. Input your table mappings in one of the following ways:
 - Click **Auto Discover**. MC retrieves the table mappings based on the database and schema mappings you provided.
 - Click **Manual Entry**. Manually input table mappings.

- Click **Load Configurations**. If you previously saved a data source configuration for the database in a file, import the file to use that configuration for the currently selected user.
8. Optionally, click **Save Configurations** to export this configuration file. You can create a mapping for another database user with this configuration file later.
 9. Click **Apply** to save and apply your configuration settings.

Reports Using Unmapped Schemas

If a report in MC needs to access a locally stored schema or table that is unmapped, MC includes information from the local DC tables for that schema to complete the report.

For remote configurations, if a report depends on an unmapped schema or table, the entire report is run against the local DC tables. If the remote database is down when MC attempts to run a report against it, MC reruns the report against the local database.

When the MC runs a report, it records missing mappings in the MC log under the INFO severity level.

Loading Data Using MC

You can use the Data Load Activity page in Management Console to import data from Amazon S3 storage to an existing table in Vertica.

MC uses the Vertica library for Amazon Web Services (AWS) to import data directly from Amazon S3 storage to Vertica. You do not need to use any third-party scripts or programs. When you run a loading job, Vertica appends rows to the target table you provide. If the job fails, or you cancel the job, Vertica commits no rows to the target table.

When you view your load history on the Instance tab, loading jobs initiated in Management Console using Amazon S3 have the name MC_S3_Load in the Stream Name column.

For more information about loading data from Amazon S3 storage to Vertica, see [Bulk Loading and Exporting Data From Amazon S3](#).

Prerequisites

To use the Load feature in Management Console, you must first have:

- Access to an Amazon S3 storage account.
- An installed version of Vertica 7.2.2 or later, which includes the Vertica library for Amazon Web Services.
- An existing table in your Vertica database to which you can copy your data. You must be the owner of the table.

Create a Loading Job

To load data from an Amazon S3 bucket to an existing table in your target database:

1. On your target database's Management Console (MC) dashboard, click the **Load** tab at the bottom of the page to view the Data Load Activity page.
2. Click the **Instance** tab.
3. Click **New S3 Data Load** at the top-right of the tab. The **Create New Amazon S3 Loading Job** dialog box opens.

4. Enter your AWS account credentials and your target location information in the required fields, which are indicated by asterisks (*).
5. (Optional) Specify additional options by completing the following fields:
 - Direct
 - COPY Parameters
 - Capture rejected data in a table
 - Reject max

For more about using these fields, see [About Configuring a Data Load from S3](#).

Cancel an Initiated Loading Job

If a loading job is in progress, you can cancel it using the **Cancel** option in the Load History tab's Cancel column. Click **Cancel** to cancel the loading job. When you cancel a job, Vertica rolls back all rows and does not commit any data to the target table.

Status	Time Started	User	Schema Name	Table Name	Execution Time (ms)	Accepted Rows	Rejected Rows	Stream Name	Cancel
Running	Mar 28, 2016 1:...	uidbadadmin	v_dbd_foo	vs_designs	0	0	N/A		Cancel
Success	Mar 28, 2016 1:...	natalia	store	store_orders_fact	10707	299994	Details	MC_S3_Load	
Failure	Mar 28, 2016 1:...	natalia	store	store_orders_fact	581	0	Details	MC_S3_Load	
Success	Mar 28, 2016 12:...	natalia	store	store_orders_fact	1299	11	N/A	S3_Data_load	
Failure	Mar 28, 2016 12:...	natalia			2	0	N/A	S3_Data_load	
Success	Mar 25, 2016 3:...	uidbadadmin	store	store_orders_fact	17990	300000	Details	MC_S3_Load	

See Also

- [Viewing Load History](#)
- [About Configuring a Data Load from S3](#)[Integrating with Apache Kafka](#)
- [Bulk Loading and Exporting Data From Amazon S3](#)
- [Loading Data From Amazon S3](#)

- [COPY](#)
- [COPY Parameters](#)

About Configuring a Data Load from S3

When you create a new S3 Data Load using MC, you have the option of further configuring the job. You can optionally specify the following:

Load Data into ROS or WOS

You can use the **Direct** field to specify whether to load the data directly into [ROS \(Read Optimized Store\)](#) containers. This is the default setting, and best for large loads of 100MB or more. For smaller loads, deselect this option to load the data into [WOS \(Write Optimized Store\)](#).

Add COPY Parameters

MC performs the loading job using a COPY statement. You can use the COPY Parameters field to further configure the COPY statement, or leave it blank. Only parameters used after the SOURCE parameter in a COPY statement are valid in this field.

Note that if you use the FILTER and PARSER parameters, they must appear in that order and precede any other parameters you use in the COPY Parameters field.

Specify EXCEPTIONS only if you set the **Capture rejected data in a table** field to No.

In the following example, you could use the DELIMITER and SKIP parameters in the COPY parameters field to separate columns with a comma and skip 1 record of input data.

```
DELIMITER ',' SKIP 1
```

To add comments in the COPY statement using this field, begin your comment with a forward slash followed by an asterisk (/*) and end it with an asterisk followed by a forward slash (*). Using a double hyphen (--) does not work in this field.

If COPY rejects the maximum number of rows, Vertica rolls back the target table rows without committing any data.

Capture Rejected Data in a Table

Use the **Capture rejected data in a table** field to create a rejected data table. A rejected data table allows you to view details about rejections through MC. If you set this field to Yes, you will be able to view rejected row data in the Load History tab.

You must have CREATE privilege on the schema if the table doesn't already exist. When you invoke multiple load processes for the same target table, MC appends all rejections data to the same table. MC generates a table using the following naming convention: *schema.s3_load_rejections_target-table-name*.

For more information about capturing rejected data, see [Saving Rejected Data To a Table](#).

Capturing Rejected Data in a Table

Use the **Capture rejected data in a table** field to create a rejected data table. A rejected data table allows you to view details about rejections through MC.

You must have CREATE privilege on the schema if the table doesn't already exist. When you invoke multiple load processes for the same target table, MC appends all rejections data to the same table. MC generates a table using the following naming convention: *schema.s3_load_rejections_target-table-name*.

For more information about capturing rejected data, see [Saving Rejected Data To a Table](#).

Set a Rejected Records Maximum

Use the Reject Max field to specify the maximum number of records that can be rejected before the load fails. If COPY rejects the maximum number of rows, Vertica rolls back the target table rows without committing any data.

Configuration Options

To further configure the loading job, you can use the following optional fields.

Field	Details
Direct	Use the Direct option to indicate whether to load the data into WOS or ROS.

Field	Details
	<ul style="list-style-type: none"> • Selected (Default): The data loads directly into ROS (Read Optimized Store) containers. Use this for large bulk loads (100MB or more). • Deselected: Data loads into WOS (Write Optimized Store). Use this for smaller bulk loads.
COPY Parameters	<p>Use the COPY Parameters field to further configure the COPY statement that will load your data.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • FILTER • PARSER • DELIMITER • TRAILING NULLCOLS • NULL • ESCAPE (NO ESCAPE) • ENCLOSED • RECORD TERMINATOR • SKIP, SKIP BYTES • TRIM 'byte' • EXCEPTIONS 'path' • ENFORCELENGTH • ABORT ON ERROR • STORAGE <p>For more information about COPY syntax, see COPY and COPY Parameters.</p>
Capture rejected data in a table	<p>Use this field to create a rejected data table.</p> <ul style="list-style-type: none"> • Yes: MC generates a rejected data table.

Field	Details
	<ul style="list-style-type: none">• No: MC does not save rejected rows to a table. <p>For more information, see Saving Rejected Data To a Table.</p>
Reject max	Use this field to specify the maximum number of records that can be rejected before the load fails.

- [Loading Data Using MC](#)
- [Viewing Load History](#)[Integrating with Apache Kafka](#)
- [Bulk Loading and Exporting Data From Amazon S3](#)
- [Loading Data From Amazon S3](#)
- [COPY](#)
- [COPY Parameters](#)

Viewing Load History

You can view a history of all your continuous and instance loading jobs in Vertica on the Data Load Activity page.

- **Continuous jobs:** Loading jobs that continuously monitor a source and stream data from the source.
- **Instance jobs:** Loading jobs that batch load from a source. Instance jobs are of a fixed length and shorter-term than continuous loads.

View Continuous Loads

The Continuous tab on the Data Load Activity page displays history of your database's continuous loading jobs. For example, you can see loading jobs you create using the Vertica integration with Kafka (see [Integrating with Apache Kafka](#)). Additionally, if you enable the MC extended monitoring feature, the Continuous tab displays the continuous jobs that stream data from your monitored database to a storage database. (See [Extended Monitoring](#) for more on how MC can use Kafka to monitor databases externally.)

Use the Continuous tab to view details about continuous jobs, such as their source, target tables, and other microbatch configuration details.

If extended monitoring is enabled, jobs streaming to the MC storage database show mc_dc_kafka_config as the scheduler name. Deselect **Show MC data collector monitoring streams** at the top of the tab to remove these jobs from the display.

In the Continuous tab, click the labels in the **Scheduler**, **Microbatch**, and **Errors Last Hour** to view additional details about those loading jobs.

Vertica Management Console dbadmin Log out

Databases and Clusters > MCStorageDB > Data Load Activity Auto Refresh Last update: 05 Nov 2016 11:17:28

Continuous **Instance**

Show MC data collector monitoring streams

Scheduler	Microbatch	Source	Target Schema	Target Table	Kafka Cluster	Timestamp Batch Start	Messages Last Batch	Messages Last Hour	Rows Accepted Last Hour	Rows Rejected Last Hour	Errors Last Hour	End Reason	Microbatch Status	Microbatch Action
							greater than less than							
↑ mc_dc...	dc_resource...	dc_resource...	dcschema	dc_resource...	mckafkaclu...	Nov 5, 2016 11:17:19 ...	0	3300	2750	0	1	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_resource...	dc_resource...	dcschema	dc_resource...	mckafkaclu...	Nov 5, 2016 11:17:20 ...	0	0	0	0	0	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_resource...	dc_resource...	dcschema	dc_resource...	mckafkaclu...	Nov 5, 2016 11:17:12 ...	1	453	403	0	1	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_session...	dc_session...	dcschema	dc_session...	mckafkaclu...	Nov 5, 2016 11:17:14 ...	12	3022	2511	0	1	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_session...	dc_session...	dcschema	dc_session...	mckafkaclu...	Nov 5, 2016 11:17:16 ...	16	3044	2515	0	0	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_spread...	dc_spread...	dcschema	dc_spread...	mckafkaclu...	Nov 5, 2016 11:17:12 ...	27	1796	1487	0	1	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_startups...	dc_startups...	dcschema	dc_startups...	mckafkaclu...	Nov 5, 2016 11:17:10 ...	0	0	0	0	0	source issue	⊕ Active	-select-
↑ mc_dc...	dc_storage...	dc_storage...	dcschema	dc_storage...	mckafkaclu...	Nov 5, 2016 11:17:15 ...	29	1740	1450	0	2	end of stream	⊕ Active	-select-
↑ mc_dc...	dc_tuning_r...	dc_tuning_r...	dcschema	dc_tuning_r...	mckafkaclu...	Nov 5, 2016 11:17:15 ...	0	0	0	0	1	end of stream	⊕ Active	-select-

For more on continuous data streaming terminology, see [Data Streaming Integration Terms](#).

View Load Instances

In the Instance tab, you can see a history of your database's one-time loading jobs. For example, you can view instance jobs you created using the COPY command in vsql (see [COPY](#)), or instance jobs you created in MC to copy data from an Amazon S3 bucket. (For more about initiating loading jobs in MC, see [Loading Data Using MC](#).)

In the Instance tab, click the labels in the Status column and Rejected Rows column to view more details about completed jobs. For more about rejected rows, see [Capturing Load Rejections and Exceptions](#).

Continuous **Instance**

Load history for the past: 1 hour New S3 Data Load

Status	Source File	Time Started	User	Schema Name	Table Name	Execution Time (ms)	Accepted Rows	Rejected Rows	Stream Name	Cancel
Success	mcqabucket/cs...	Nov 4, 2016 5:34:05 PM	fred	public	townsfile	277	234	4	MC_S3_Load	
Success	mcqabucket/cs...	Nov 4, 2016 5:31:43 PM	fred	public	townsfile	2286	238	0	MC_S3_Load	

The number of load history results on the Instance tab depends on the [Data Collector](#) retention policy for Requests Issued and Requests Completed. To change the retention policy, see [Configuring Data Retention Policies](#).

See Also

- [Loading Data Using MC](#)
- [Integrating with Apache Kafka](#)
- [Integrating with Apache Kafka](#)
- [Bulk Loading and Exporting Data From Amazon S3](#)
- [Loading Data From Amazon S3](#)
- [COPY](#)
- [COPY Parameters](#)

Viewing Profile Data in MC

Management Console allows you to view profile data about a single query. You can:

- Review the profile data in multiple views
- View details about projection metadata, execution events, and optimizer events
- Identify how much time was spent in each phase of query execution and which phases took the most amount of time

After you select the database you want to use, you can view the profile data using Management Console in either of two ways:

- Focus on specific areas of database activity, such as spikes in CPU usage
- Review the profile data for a specific query

To focus on specific areas of database activity:

1. At the bottom of the Management Console window, click the **Activity** tab.
2. From the list at the top of the page, select **Queries**.
3. On the activity graph, click the data point that corresponds to the query you want to view.
4. In the **View Plan** column, click **Profile** next to the command for which you want to view the query plan. Only certain queries, like SELECT, INSERT, UPDATE, and DELETE, have profile data.
5. In The **Explain Plan** window, Vertica profiles the query.
6. You can view the output in Path Information view, Query Plan Drilldown view, Tree Path view, or Profile Analysis view. To do so, click the respective buttons on the left of the output box.

To review the profile data for a specific query:

1. In the **Explain** window, type or paste the query text into the text box. Additionally, you can monitor queries that are currently running. To do so, perform one of the following. In the **Find a Query By ID** input window:
 - Enter the query statement and transaction ID
 - Click the **Browse Running Queries** link

Caution: If you enter more than one query, Management Console profiles only the first query.

2. To receive periodic updates about the query's progress and resources used, select the **Enable Monitoring** check box. As a best practice, avoid specifying an interval time of less than 60 seconds because doing so may slow your query's progress.
3. Click the **Profile** button.

While Vertica is profiling the query, a **Cancel Query** button is enabled briefly, allowing you to cancel the query and profiling task. If the **Cancel Query** button is disabled, that means Management Console does not have the proper information to cancel the query or the query is no longer running in the database.

When processing completes, the profile data and metrics display below the text box. You can view the output in Path Information view, Query Plan Drilldown view, Tree Path view, or Profile Analysis view. To do so, click the respective view buttons on the left of the output box.

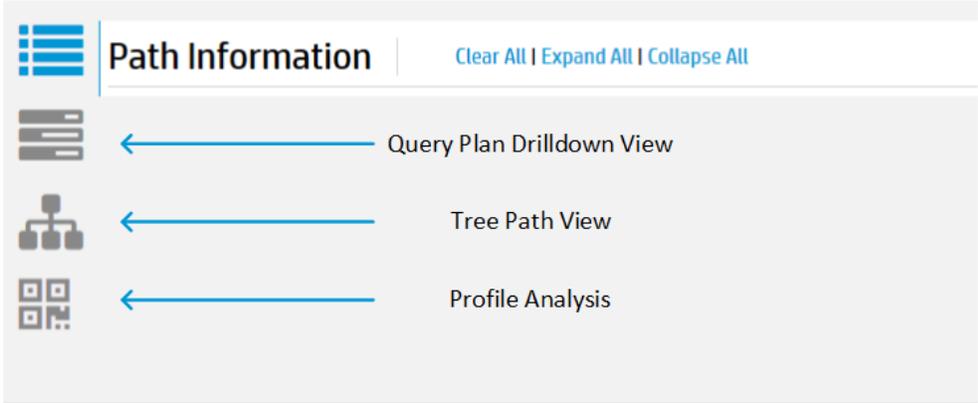
Viewing Different Profile Outputs

Vertica Management Console allows you to examine the results of your query profile in multiple views. You can view your profile in the following formats:

- Path Information view
- Query Drilldown view

- Tree Path view
- Profile Analysis view

You can change the query profile output using the icons on the bottom portion of the **Explain** page.

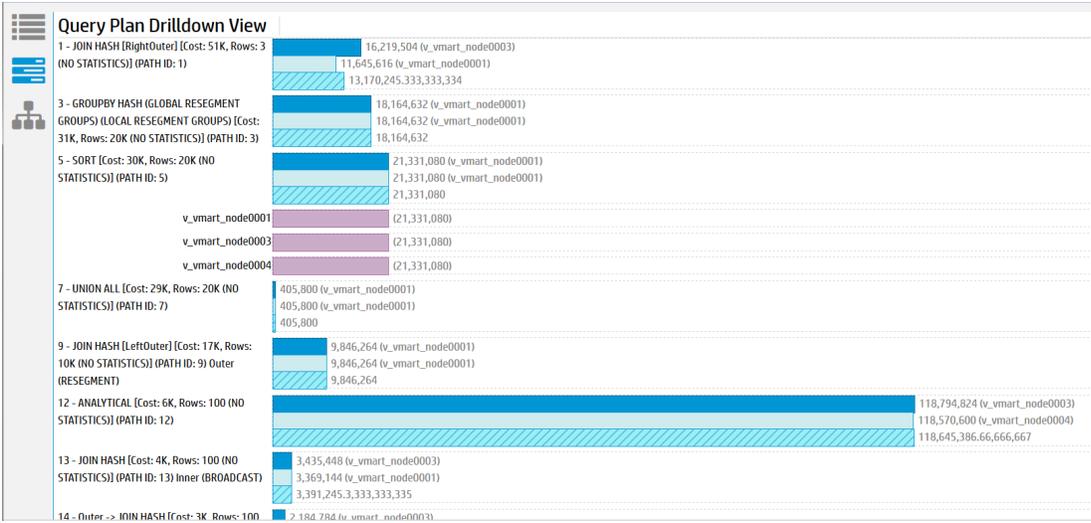


The **Path Information** view displays the query plan path along with metric data. If you enable profile monitoring, the data will update at the specified interval. To view metadata for a projection or a column, click the object name in the path output. A pop-up window displays the metadata if it is available.

The screenshot shows a detailed view of the 'Path Information' section. On the left, a tree view shows the query plan structure with nodes like JOIN HASH, GROUPBY HASH, SORT, UNION ALL, and ANALYTICAL. On the right, a table displays performance metrics for each node. The table has columns for Disk, Memory, Sent, Received, and Time, each with a progress bar. The ANALYTICAL node shows significant activity in the Sent, Received, and Time columns.

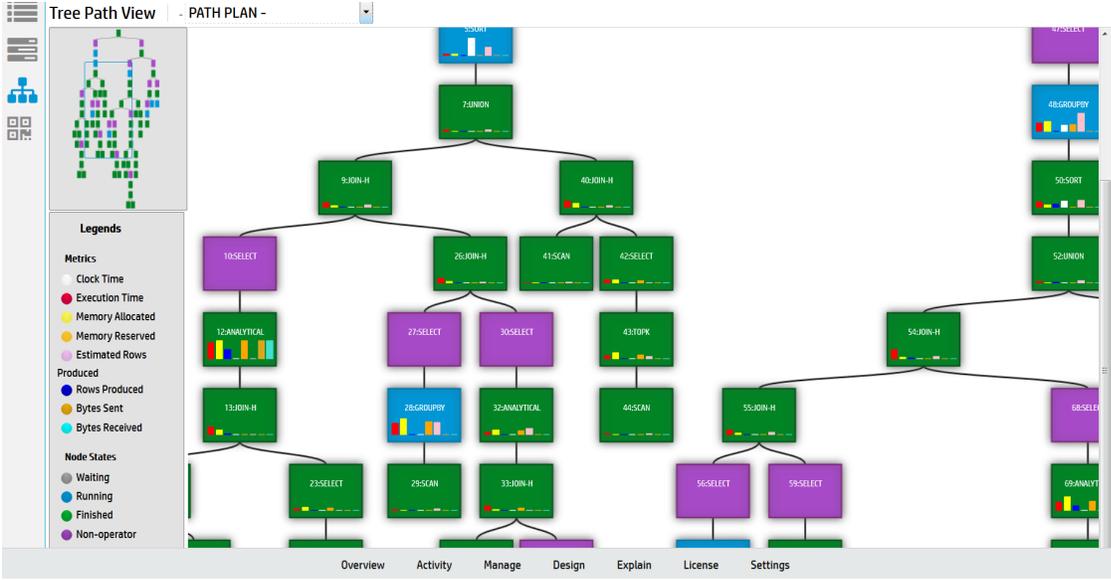
Node / Operator	Disk	Memory	Sent	Received	Time
+ JOIN HASH [RightOuter] [Cost: 51K, Rows: 3 (NO STATISTICS)] (PATH ID: 1)	Progress bar				
+----> GROUPBY HASH (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 31K, Rows: 20K (NO STATISTICS)] (PATH ID: 3)	Progress bar				
+----> SORT [Cost: 30K, Rows: 20K (NO STATISTICS)] (PATH ID: 5)	Progress bar				
+----> UNION ALL [Cost: 29K, Rows: 20K (NO STATISTICS)] (PATH ID: 7)	Progress bar				
+----> JOIN HASH [LeftOuter] [Cost: 17K, Rows: 10K (NO STATISTICS)] (PATH ID: 9) Outer (RESEGMENT)	Progress bar				
+----> ANALYTICAL [Cost: 6K, Rows: 100 (NO STATISTICS)] (PATH ID: 12)	Progress bar				

The **Query Plan Drilldown** view shows detailed counter information at the node and operator level.

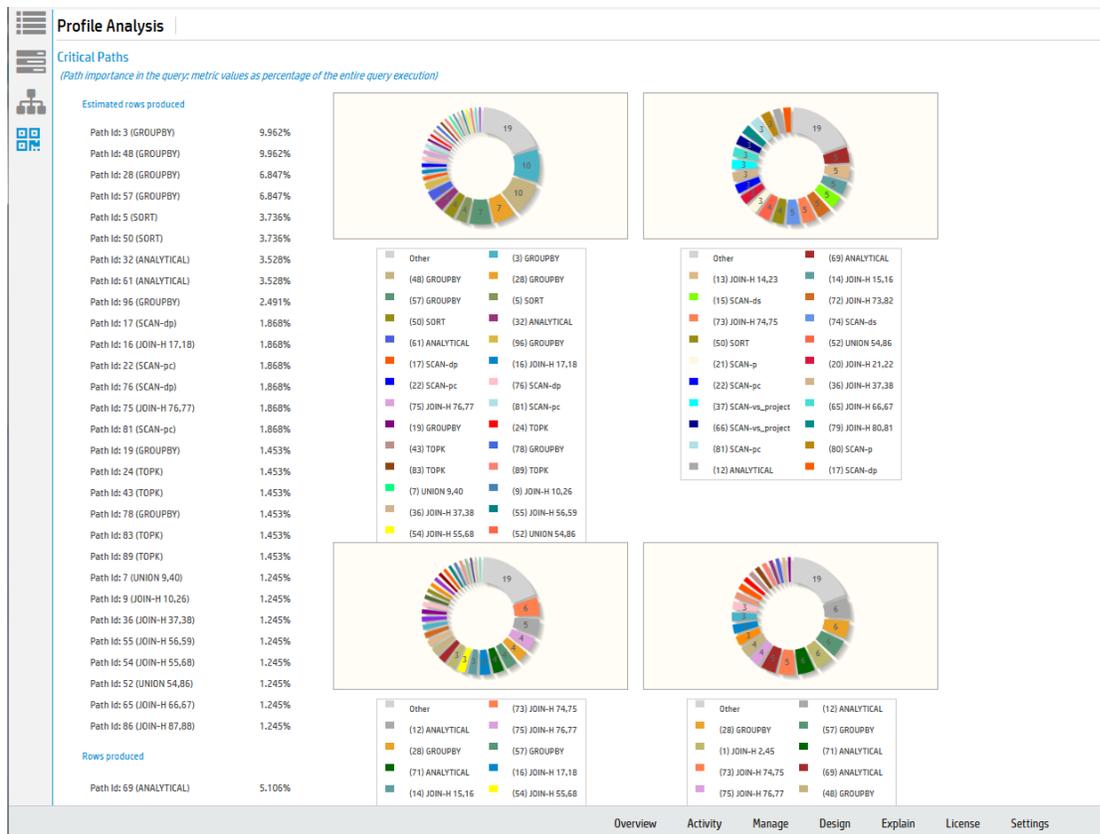


For each path, the path number is listed along with statistical information on the node and operator level. This view allows you to see which nodes are acting as outliers. Click on any of the bars to expand details for that node.

The **Tree Path** details the query plan in the form of a tree. If you enable monitoring, the state of the path blocks will change depending on whether the path is running, done, or has not started. Metric information is displayed in each path block for the counters you specified in the Profile Settings.



The **Profile Analysis** view allows you to identify any resource outliers. You can compare the estimated rows produced count with the actual rows produced count, view execution time per path, and identify memory usage per path.



When you profile a query, you will also see a pie chart detailing the query phase duration. You can also view projection metadata, execution events, and optimizer events by clicking on the respective buttons next to the pie chart.

Monitoring Profiling Progress

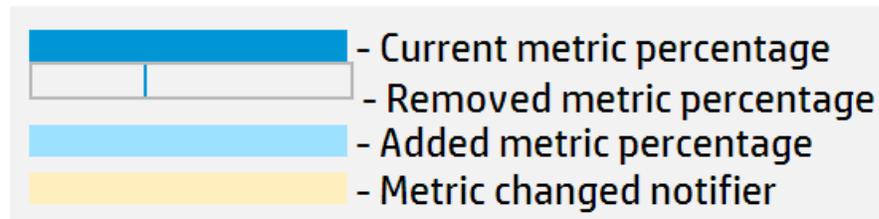
While loading profile data for a query, Management Console can provide updates about the query's progress and resources used.

To enable profiling progress updates, select the Enable Monitoring check box when profiling a query. See [Viewing Profile Data in Management Console](#).

The default interval time is 60 seconds. At the specified interval, Management Console displays an updated view of the query's progress. Note that interval times of less than 60 seconds may slow down your query.

Viewing Updated Profile Metrics

At every interval, Management Console displays a new set of profile metrics. You can view these metrics in Path Information view, Query Plan Drilldown view, or Tree view by clicking the respective view buttons on the left of the output box.



- A dark blue bar indicates the current metric percentage.
- When a metric bar has decreased, a dark blue line indicates the previous metric percentage.
- When a metric bar has increased, a light blue bar indicates the added percentage. The previous percentage appears as a dark blue bar.
- A metric bar highlighted in yellow indicates it has changed since the last interval.
- A metric bar highlighted in red indicates the absolute value of the metric has decreased. This typically means Vertica reported the previous value incorrectly, and has readjusted. (For example, if Vertica previously reported path's Time value as 75 seconds, then reports it as 50 seconds at the next interval, the metric bar turns red to indicate the decrease in absolute Time value.)

Expanding and Collapsing Query Path Profile Data

When you have a query on the EXPLAIN window, the profile data displays in the right-hand side of the lower half of the window. The query path information can be lengthy, so you can collapse path information that is uninteresting, or expand paths that you want to focus on.

- To collapse all the query paths, click **Collapse All**.
- To expand all the query paths, click **Expand All**.
- To expand an individual query path so you can see details about that step in processing the query, click the first line of the path information. Click the first line again to collapse the path data.

For information about what the profile data means, see [About Profile Data in Management Console](#).

About Profile Data in Management Console

After you profile a specific query, the Management Console Explain page displays profile data like query duration, projection metadata, execution events, optimizer events, and metrics in a pie chart.

See the following links for more information on the kinds of profile data you can review on the Management Console Explain page:

- [Projection Metadata](#)
- [Query Phase Duration](#)
- [Profile Metrics](#)
- [Execution Events](#)
- [Optimizer Events](#)

Projection Metadata

To view projection metadata for a specific projection, click the projection name in the EXPLAIN output. Metadata for that projection opens in a pop-up window.

To view projection data for all projections accessed by that query, click the **View Projection Metadata** button at the top of the **Explain** page. The metadata for all projections opens in a new browser window.

Note: If the **View Projection Metadata** button is not enabled, click **Profile** to retrieve the profile data, including the projection metadata.

The projection metadata includes the following information:

- Projection ID
- Schema name
- Whether or not it is a superprojection
- Sort columns
- IDs of the nodes the projection is stored on
- Whether or not it is segmented
- Whether or not it is up to date
- Whether or not it has statistics
- Owner name
- Anchor table name

To display a SQL script that can recreate the projection on a different cluster, click **Click to get export data**. This script is identical to the output of the [EXPORT_OBJECTS](#) function. The SQL script opens in a pop-up window.

Copy and paste the command from this window, and click **Close**.

Query Phase Duration

This pie chart appears in the upper-right corner of the Query Plan window. It shows what percentage of total query processing was spent in each phase of processing the query.

The phases included in the pie chart (when applicable) are:

- Plan
- InitPlan
- SerializePlan
- PopulateVirtualProjection
- PreparePlan
- CompilePlan

- ExecutePlan
- AbandonPlan

Hover over the slices on the pie chart or over the names of the phases in the box to get additional information. You can see the approximate number of milliseconds (ms) and percentage used during each phase.

Note: The time data in the profile metrics might not match the times in the query phase duration. These times can differ because the query phase duration graph uses the longest execution time for a given phase from all the nodes. Network latency can add more data, which is not taken into account in these calculations.

Profile Metrics

In the **Path Information** view, the area to the right of each query path contains profile metrics for that path.

- **Disk**—Bytes of data accessed from disk by each query path. If none of the query paths accessed the disk data, all the values are 0.
- **Memory**—Bytes of data accessed from memory by each query path.
- **Sent**—Bytes of data sent across the cluster by each query path.
- **Received**—Bytes of data received across the cluster by each query path.
- **Time**—Number of milliseconds (ms) that the query path took to process on a given node, shown on progress bars. The sum of this data does not match the total time required to execute the query. This mismatch occurs because many tasks are executed in parallel on different nodes.

Hover over the progress bars to get more information, such as total bytes and percentages.

Note: The time data in the profile metrics might not match the times in the [Query Phase Duration](#) chart. These times can differ because the query phase duration graph uses the longest execution time for a given phase from all the nodes. Network latency can add more data, which is not taken into account in these calculations.

Execution Events

To help you monitor your database system, Vertica logs significant events that affect database performance and functionality. Click **View Execution Events** to see information about the events that took place while the query was executing.

If the **View Execution Events** button is not enabled, click **Profile** to retrieve the profile data, including the execution events.

The arrows on the header of each column allow you to sort the table in ascending or descending order of that column.

The execution events are described in the following table.

Event Characteristic	Details
Time	Clock time when the event took place.
Node Name	Name of the node for which information is listed.
Session ID	Identifier of the session for which profile information is captured.
User ID	Identifier of the user who initiated the query.
Request ID	Unique identifier of the query request in the user session.
Event Type	Type of event processed by the execution engine. For a list of events and their descriptions, see Initial Process for Improving Query Performance .
Event Description	Generic description of the event.
Operator Name	Name of the Execution Engine component that generated the event. Examples include but are not limited to: <ul style="list-style-type: none">• DataSource• DataTarget• NetworkSend• NetworkRecv• StorageUnion

	Values from the Operator name and Path ID columns let you tie a query event back to a particular operator in the query plan. If the event did not come from a specific operator, then this column is NULL.
Path ID	Unique identifier that Vertica assigns to a query operation or a path in a query plan. If the event did not come from a specific operator, this column is NULL.
Event OID	A unique ID that identifies the specific event.
Event Details	A brief description of the event and details pertinent to the specific situation.
Suggested Action	Recommended actions (if any) to improve query processing.

Optimizer Events

To help you monitor your database system, Vertica logs significant events that affect database performance and functionality. Click **View Optimizer Events** to see a table of the events that took place while the optimizer was planning the query.

If the **View Optimizer Events** button is not enabled, click **Profile** to retrieve the profile data, including the optimizer events.

The arrows on the header of each column allow you to sort the table in ascending or descending order of that column.

The following types of optimizer events may appear in the table:

Event characteristic	Details
Time	Clock time when the event took place.
Node Name	Name of the node for which information is listed.
Session ID	Identifier of the session for which profile information is captured.
User ID	Identifier of the user who initiated the query.
Request ID	Unique identifier of the query request in the user session.
Event Type	Type of event processed by the optimizer.

Event Description	Generic description of the event.
Event OID	A unique ID that identifies the specific event.
Event Details	A brief description of the event and details pertinent to the specific situation.
Suggested Action	Recommended actions (if any) to improve query processing.

Clearing Query Data

After you finish reviewing the current query data, click **Clear All** to clear the query text and data. Alternatively, to display information about another query, enter the query text and click **Explain** or **Profile**.

Extended Monitoring

Enabling extended monitoring allows you to monitor a longer range of data through MC. This can offer insight into long-term trends in your database's health. MC can also continue to display your monitored database's dashboard while it is down.

Extended monitoring uses Kafka to stream monitoring data from your monitored databases to a single MC storage database. MC can query the storage database instead of your monitored database to render some of its charts, reducing impact on your monitored database's performance.

How Extended Monitoring Works

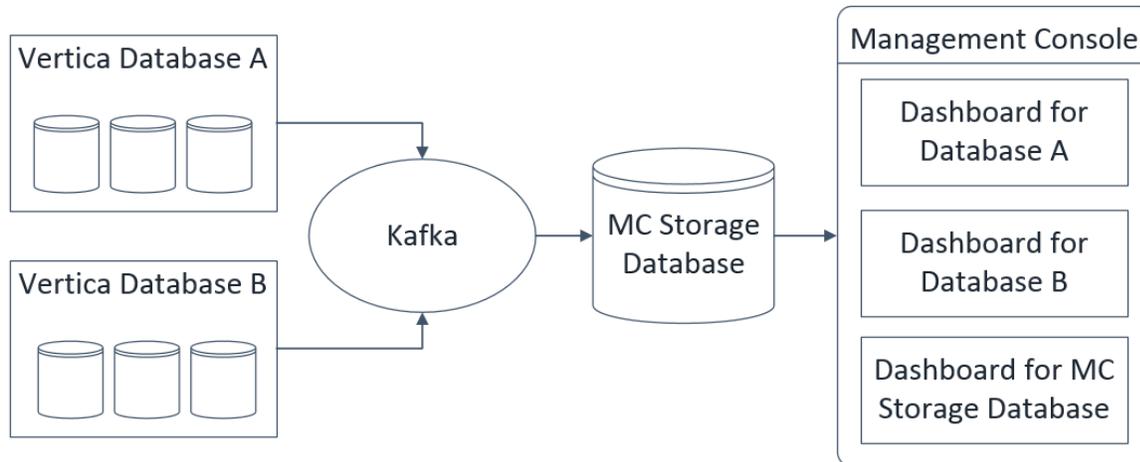
By default, MC monitors your database by querying it directly for monitoring data about system activities, performance, and resource utilization. Typically, the [Data Collector](#) stores all monitoring data in data collector (DC) tables. However, DC tables have limited retention periods. See [Retaining Monitoring Information](#).

Extended monitoring stores your database's monitoring data in a dedicated storage database. Vertica streams data from your database's DC tables through Kafka servers to the storage database. To use extended monitoring, you must have access to a running Kafka server. For more how Vertica integrates with Kafka, see [Integrating with Apache Kafka](#).

After you set up and enable extended monitoring for a monitored database, MC renders several of your database's charts and graphs by querying the MC storage database instead of directly querying the database you are monitoring.

You can enable extended monitoring for any, or all, of your monitored databases. The MC storage database provides a single repository for monitoring data from every database that uses enabled extended monitoring.

In the following example, Kafka streams system data from two monitored databases to the storage database. MC uses the storage database to render individual dashboards for each monitored database. Be aware that MC always creates a dashboard that monitors the MC storage database.



Use Extended Monitoring

Important: To use extended monitoring, Micro Focus recommends installing Management Console on a host without any other Vertica databases.

When a database has extended monitoring enabled, the MC charts that use the feature display a rocket ship icon in the corner. You can use these charts to access longer-term data about your database's health or performance.

To view historical information in these charts, click the calendar icon to specify the timeframe to display. For example, if your database has been down for several hours, your charts do not display recent activity in your database. You could use the timeframe filter in the System Bottlenecks chart to see unusual resource usage occurred in your database in the hour it went down.

You can view a history of the Kafka streaming jobs loading data into the storage database. MC displays these jobs on the Load tab of your storage database's dashboard. See [Viewing Load History](#).

Set Up Extended Monitoring

To set up extended monitoring, see [Managing the Storage Database](#) and [Managing Extended Monitoring on a Database](#).

See Also

- [Managing the Storage Database](#)
- [Managing Extended Monitoring on a Database](#)
- [Viewing Load History](#)
- [Integrating with Apache Kafka](#)
- [Retaining Monitoring Information](#)

Managing the Storage Database

Extended Monitoring stores your Vertica database's monitoring data in a dedicated MC storage database.

To use Extended Monitoring, you must first set up the storage database and configure it for Kafka streaming. Then, turn on Extended Monitoring for any or all monitored databases.

MC automatically configures a schema for the storage database, named `dcschema`, which is synced with DC tables on your monitored databases.

Caution: Do not alter `dcschema` after MC has configured it. Altering it could cause the storage database to lose data or supply incorrect monitoring information to MC.

MC Preparation

First verify that MC is not installed on the same host as a Vertica database. When Extended Monitoring is enabled, MC sharing a host with a production database can affect performance.

You must also increase the allocation of memory for the MC application server. See [Modify Memory Allocation](#). Tune these options based on:

- The demands of your database
- The amount of monitoring data you plan to view in MC charts at once.

For example, MC requires more memory to display a week of data in a chart.

Modify Memory Allocation

To modify memory allocation:

1. In Management Console, select the **Configuration** tab on the MC Settings page.
2. Modify the following fields under **Application Server JVM Settings** to increase the allocation of memory for the JVM:
 - **Initial Heap Size:** For Extended Monitoring, a minimum value of 2GB is recommended. (The default value is 1GB.)
 - **Maximum Heap Size:** For Extended Monitoring, a minimum value of 4GB is recommended. (The default value is 2GB.)
3. Click **Apply** at the top right of the page. A prompt appears to restart MC.
4. Click **OK** to restart MC and save your changes.

Storage Database Requirements

To set up storage for Extended Monitoring, your system must meet the following prerequisites:

- Vertica 8.x
- An available host, or available database whose Vertica version corresponds to the version of the database you plan to monitor. (eg. To use Extended Monitoring for a Vertica 8.1.0 database, its storage database should also be on Vertica 8.1.0.)
- Configured MC for Extended Monitoring (See [MC Preparation](#))
- Access to a deployed Kafka server (For details on installing Kafka, see the [Apache Kafka site](#))

Set Up the Storage Database

To set up the storage database for Extended Monitoring:

1. On the MC Settings page, select the MC Storage DB Setup tab.
2. In the Server field in the Kafka Broker section, enter the host name or IP addresses and ports for one or more of your deployed Kafka servers.
3. Designate the storage database in the MC External Storage Database section. You can create a new database, or use an existing database.
 - **Create a new database:** Choose this option to create a new single node cluster on an available host using a Community Edition license of Vertica. This choice does not affect your normal Vertica license usage.
 - **Use an existing database known to MC:** Choose this option to designate a database you have already imported to MC. If the schema 'dcschema' exists in the database, a dialog appears:
 - Click **Append** to keep the existing schema's data. For example, if you have already used this database for Extended Monitoring storage and are re-importing it, you can use this option to retain its historical data for continued use.
 - Click **Remove** to clear the existing schema from the database, and creates a fresh version of dcschema configured for Extended Monitoring storage.
4. Click **Enable Streaming** to enable data streaming from the Kafka server to the MC storage database.
5. Turn on Extended Monitoring for your databases on the Extended Monitoring tab. See [Managing Extended Monitoring on a Database](#) for more information.

Restart the Storage Database

If you stop the storage database while streaming is enabled, streaming to the storage database stops automatically. You must re-enable streaming on the MC Storage DB Setup tab after you restart the storage database.

If streaming to the MC storage database is disabled while Extended Monitoring on your database is on, the Kafka retention policy determines how long streaming can remain disabled without data loss. See [Managing Streaming Services for Extended Monitoring](#).

Discontinue the Storage Database

1. Select the Extended Monitoring tab in MC Settings.
2. Set Extended Monitoring for all databases to **OFF**.
3. Select the MC Storage DB Setup tab in MC Settings.
4. Click **Disable Streaming** in the MC External Storage Database section to de-activate your storage database.
5. Click **Remove** in the MC External Storage Database section to remove the MC Storage Database from MC.
6. Choose whether to keep or remove the data your storage database has collected:
 - **Keep Data:** Existing data will not be removed. If you re-use this database for Extended Monitoring storage, you can choose to append new collected monitoring data to this existing data.
 - **Remove Data:** MC deletes its customized storage schema from the database.

Configure Storage Database Memory Usage

On the Resource Pools tab of the storage database, you can optionally increase the memory size of `SYSQUERY` and `KAFKA_DEFAULT_POOL`. For setting resource pool parameters in MC, see [Configuring Resource Pools in Management Console](#).

- **SYSQUERY:** Reserved for temporary storage of intermediate results of queries against system monitoring and catalog tables. Default setting is 64M. For best performance for MC, set to 2G or higher.
- **KAFKA_DEFAULT_POOL:** Reserved for all queries executed by the Kafka scheduler. Default setting is 30%, which is the recommended setting. By default, queries spill to the general pool when they exceed the 30% memory size.

Manage Disk Space

The storage database uses a customized schema, named `dcschema`. You can monitor these tables on MC, using the Table Utilization chart on the storage database's Activity tab. The Table

Utilization chart lists all the tables in dcschema and their details, such as row counts and column properties. You can sort by row count to determine if certain tables use more disk space on your storage database. See [Monitoring Table Utilization and Projections](#).

You should regularly drop partitions from dcschema if you have limited disk space for the MC storage database. MC does not automatically drop partitions from the storage database. For more information on dropping partitions, see [Dropping Partitions](#).

The table dc_execution_engine_profiles is partitioned by day. Because this table typically contains the most rows, as a best practice you should drop partitions from this table more often. The following example shows how you can specify the partition key 2016-08-22 to drop a partition from dc_execution_engine_profiles.

```
SELECT DROP_PARTITION('dcschema.dc_execution_engine_profiles',  
2016-08-22);
```

Other than dc_execution_engine_profiles, all other tables in dcschema are partitioned by week. The next example shows you how you can drop a partition from the table dc_cpu_aggregate_by_minute, specifying the thirty-fourth week of 2016.

```
SELECT DROP_PARTITION('dcschema.dc_cpu_aggregate_by_minute',  
201634);
```

Manage Client Sessions

By default Vertica allows 50 client sessions and an additional five administrator sessions per node. If you reach the limit on the storage database, MC switches back to default monitoring, and does not use Extended Monitoring data from the storage database.

You can optionally configure the maximum number of client sessions that can run on a single database cluster node on your MC storage database's Settings page:

1. On the storage database dashboard, click the **Settings** page.
2. Choose the **General** tab.
3. Input a value in the **Maximum client sessions** field. Valid values for the parameter are 0 to 1000.

For more details about managing client connections in MC, see [Managing Client Connections](#).

See Also

- [Extended Monitoring](#)
- [Managing Extended Monitoring on a Database](#)
- [Viewing Load History](#)

Managing Extended Monitoring on a Database

When you enable extended monitoring on your Vertica database, monitoring data from your database streams through Kafka servers to the MC storage database.

You can enable streaming for any or all databases that MC monitors.

Extended Monitoring Prerequisites

Before you can enable extended monitoring, your system must meet these prerequisites:

- Vertica 8.x
- Deployed Kafka server(s)
- Configured MC for extended monitoring (See [Managing the Storage Database](#))
- Deployed MC storage database (See [Managing the Storage Database](#))

Enable Extended Monitoring

1. Select the Extended Monitoring tab on MC Settings.

The Extended Monitoring page displays all databases monitored by MC.

2. In the Memory Limit field for the database of your choice, set the maximum amount of memory the database can use for streaming monitoring data. For more about the memory limit, see [Managing Streaming Services for Extended Monitoring](#).

3. In the Extended Monitoring column, select **ON** to enable streaming for the database of your choice.

The database begins streaming its monitoring data to the Kafka server.

User Access

When you change user permissions for a database using extended monitoring, the user access policy on the storage database does not automatically update. On the Extended Monitoring page, in the user access column for your database, click Refresh to sync the policy.

If you rename a Vertica user, you must re-map the user in MC Settings before refreshing the user access policy.

See Also

- [Extended Monitoring](#)
- [Managing the Storage Database](#)
- [Viewing Load History](#)
- [Integrating with Apache Kafka](#)

Managing Streaming Services for Extended Monitoring

When extended monitoring is enabled, Vertica streams data from your database through Kafka servers to the storage database.

For additional parameters that optimize the performance of Kafka with Vertica, see [Configuring Kafka for Vertica](#).

View Streaming Details in MC

Click the Load tab on your database's MC dashboard to see the Data Load Activity page. On this page, the Continuous tab displays details about all continuous loading jobs for extended monitoring. You can use this page to monitor whether your extended monitoring data is streaming successfully to the MC storage database.

See [Viewing Load History](#) for more about the Data Load Activity page.

Tip: If you do not see loading jobs for extended monitoring, verify that you have selected **Show MC data collector monitoring streams** at the top of the Continuous tab.

Prevent Data Loss

The Memory Limit buffer allows you to restart the Kafka server without data loss. Vertica queues the streamed data until you restart the Kafka server. When the Kafka server remains down for an extended period of time, data loss occurs when the queue of streamed data exceeds the buffer. You set the buffer size on the Extended Monitoring tab when you enable extended monitoring for a database. See [Managing Extended Monitoring on a Database](#).

The Kafka retention policy determines when data loss occurs during the following scenarios:

- Restarting the MC storage database (see [Managing the Storage Database](#))
- Disabling streaming on the MC storage database (see [Managing the Storage Database](#))
- Restart a micro-batch (see [Loading Data Using MC](#))

The Kafka retention policy can allow you to restart these extended monitoring components without data loss. The Kafka server retains the data while the listed components are disabled. Data loss occurs when the streamed data exceeds the Kafka retention policy's log size or retention time limits. See the [Apache Kafka documentation](#) for how to configure the retention policy.

Changing the Kafka Server

Be aware that when you change Kafka servers for extended monitoring on the MC Storage DB Setup page, you must disable all extended monitoring processes and re-configure the

MC storage database. For storage database setup instructions, see [Managing the Storage Database](#).

See Also

- [Managing Extended Monitoring on a Database](#)
- [Viewing Load History](#)
- [Configuring Kafka for Vertica](#)
- [Integrating with Apache Kafka](#)

SQL Reference Manual

Welcome to the Vertica SQL Reference Manual. This guide provides an overview of Vertica Structured Query Language (SQL). It defines system limits, describes SQL Language elements and system tables. This guide also provides reference descriptions of SQL Data Types, SQL Functions, and SQL Statements.

This document assumes that you are familiar with the basic concepts and terminology of the SQL language and relational database management systems.

SQL in Vertica

Vertica offers a robust set of SQL elements that allow you to manage and analyze massive volumes of data quickly and reliably. Vertica uses the following:

[SQL language elements](#), including:

- Keywords and Reserved Words
- Identifiers
- Literals
- Operators
- Expressions
- Predicates
- Hints

[SQL data types](#), including:

- Binary
- Boolean
- Character
- Date/Time
- Long
- Numeric

[SQL functions](#) including Vertica-specific functions that take advantage of Vertica's unique column-store architecture. For example, call [ANALYZE_STATISTICS](#) to collect and aggregate a variable amount of sample data for statistical analysis.

[SQL statements](#) that let you write robust queries to quickly return large volumes of data.

System Limits

This section describes the system limits on the size and number of objects in a Vertica database. In most cases, computer memory and disk drive are the limiting factors.

Item	Limit
Number of nodes	Maximum 128 (without Vertica assistance)
Database size	Approximates the number of files times the file size on a platform, depending on the maximum disk configuration
Table size	The smaller of: <ul style="list-style-type: none"> • 2^{64} rows per node • 2^{63} bytes per column
Row size	32,768,000 bytes Row size is approximately the sum of its maximum column sizes, where, for example, a VARCHAR(80) has a maximum size of 80 bytes.
Key size	Limited only by row size
Number of tables/projections per database	Limited by physical RAM, as the catalog must fit in memory.
Number of concurrent connections per node	Default of 50, limited by physical RAM (or threads per process), typically 1024
Number of concurrent connections per cluster	Limited by physical RAM of a single node (or threads per process), typically 1024
Number of columns per table	1600
Number of rows per load	2^{63}
Number of partitions	1024

Item	Limit
	For details, see Partitioning Best Practices in the Administrator's Guide.
Length for a fixed-length column	65000 bytes
Length for a variable-length column	32000000 bytes
Length of basic names	128 bytes. Basic names include table names, column names, etc.
Query length	No limit
Depth of nesting subqueries	Unlimited in FROM, WHERE, and HAVING clauses

SQL Language Elements

This chapter presents detailed descriptions of the language elements and conventions of Vertica SQL.

Keywords

Keywords are words that have a specific meaning in the SQL language. Every SQL statement contains one or more keywords. Although SQL is not case-sensitive with respect to keywords, they are generally shown in uppercase letters throughout this documentation for readability purposes.

Note: If you use a keyword as the name of an identifier or an alias in your SQL statements, you may have to qualify the keyword with AS or double-quotes. Vertica requires AS or double-quotes for certain reserved and non-reserved words to prevent confusion with expression syntax, or where the use of a word would be ambiguous.

Reserved Words and Keywords

Many keywords are also reserved words.

Vertica recommends that you not use reserved words as names for objects, or as identifiers. Including reserved words can make your SQL statements confusing. Where you need to use reserved words as names for objects or as identifiers, you need to double-quote them. An example of where you might need to do this is where Vertica has added a reserved word that you already use throughout your existing SQL scripts.

Note: All reserved words are also keywords, but Vertica can add reserved words that are not keywords. A reserved word can simply be a word that is reserved for future use.

Non-reserved Keywords

Non-reserved keywords have a special meaning in some contexts, but can be used as identifiers in others. You can use non-reserved keywords as aliases—for example, SOURCE:

```
=> SELECT my_node AS SOURCE FROM nodes;
```

Note: Vertica uses several non-reserved keywords in [directed queries](#) to specify special join types. You can use these keywords as table aliases only if they are double-quoted; otherwise, double-quotes can be omitted:

- ANTI
- NULLAWARE
- SEMI
- SEMIALL
- UNI

Viewing the List of Reserved and Non-reserved Keywords

To view the current list of Vertica reserved and non-reserved words, query the [KEYWORDS](#) system table.

```
VMart=> select * from keywords;
```

The output lists keywords alphabetically and identifies them as reserved (R) or non-reserved (N).

Identifiers

Identifiers (names) of objects such as schema, table, projection, column names, and so on, can be up to 128 bytes in length.

Unquoted Identifiers

Unquoted SQL identifiers must begin with one of the following:

- Letters (use A–Z or a-z only; behavior for Unicode is undefined)
- Underscore (`_`)

Subsequent characters in an identifier can be:

- Letters (as above)
- Unicode letters (letters with diacriticals or not in the Latin alphabet)
- Underscore (_)
- Digits(0–9)
- Dollar sign (\$)

Caution: The SQL standard does not support dollar sign in identifiers so usage can cause application portability problems.

Quoted Identifiers

Identifiers enclosed in double quote (") characters can contain any character. If you want to include a double quote, you need a pair of them; for example """". You can use names that would otherwise be invalid, such as names that include only numeric characters ("123") or contain space characters, punctuation marks, keywords, and so on; for example:

```
CREATE SEQUENCE "my sequence!";
```

Double quotes are required for non-alphanumerics and SQL keywords such as "1time", "Next week" and "Select".

Note: Identifiers are not case-sensitive. Thus, identifiers "ABC", "ABc", and "aBc" are synonymous, as are ABC, ABc, and aBc.

Non-ASCII Characters

Vertica accepts non-ASCII UTF-8 Unicode characters for table names, column names, and other [Identifiers](#), extending the cases in which upper/lower case distinctions are ignored (case-folded) to all alphabets, including Latin, Cyrillic, and Greek.

For example, the following CREATE TABLE statement uses the ß (German eszett) in the table name:

```
=> CREATE TABLE straÙe(x int, y int);  
CREATE TABLE
```

Identifiers Are Stored As Created

SQL identifiers, such as table and column names, are no longer converted to lowercase. They are stored as created, and references to them are resolved using case-insensitive compares. It is not necessary to double quote mixed-case identifiers. For example, the following statement creates table ALLCAPS.

```
=> CREATE TABLE ALLCAPS(c1 varchar(30));  
=> INSERT INTO ALLCAPS values('upper case');
```

The following statements are variations of the same query and all return identical results:

```
=> SELECT * FROM ALLCAPS;  
=> SELECT * FROM allcaps;  
=> SELECT * FROM "allcaps";
```

All three commands return the same result:

```
      c1  
-----  
upper case  
(1 row)
```

Note that the system returns an error if you try to create table AllCaps:

```
=> CREATE TABLE AllCaps(c1 varchar(30));  
ROLLBACK: table "AllCaps" already exists
```

See [QUOTE_IDENT](#) for additional information.

Literals

Literals are numbers or strings used in SQL as constants. Literals are included in the select-list, along with expressions and built-in functions and can also be constants.

Vertica provides support for number-type literals (integers and numerics), string literals, VARBINARY string literals, and date/time literals. The various string literal formats are discussed in this section.

Number-Type Literals

Vertica supports three types of numbers: integers, numerics, and floats.

- **Integers** are whole numbers less than 2^{63} and must be digits.
- **Numerics** are whole numbers larger than 2^{63} or that include a decimal point with a precision and a scale. Numerics can contain exponents. Numbers that begin with 0x are hexadecimal numerics.

Numeric-type values can also be generated using casts from character strings. This is a more general syntax. See the Examples section below, as well as [Data Type Coercion Operators \(CAST\)](#).

Syntax

```
digits  
digits.[digits] | [digits].digits  
digits e[+/-]digits | [digits].digits e[+/-]digits | digits.[digits] e[+/-]digits
```

Parameters

<i>digits</i>	One or more numeric characters, 0 through 9
<i>e</i>	Exponent marker

Notes

- At least one digit must follow the exponent marker (e), if e is present.
- There cannot be any spaces or other characters embedded in the constant.
- Leading plus (+) or minus (–) signs are not considered part of the constant; they are unary operators applied to the constant.
- In most cases a numeric-type constant is automatically coerced to the most appropriate type depending on context. When necessary, you can force a numeric value to be interpreted as a specific data type by casting it as described in [Data Type Coercion Operators \(CAST\)](#).
- Floating point literals are not supported. If you specifically need to specify a float, you can cast as described in [Data Type Coercion Operators \(CAST\)](#).
- Vertica follows the IEEE specification for floating point, including NaN (not a number) and Infinity (Inf).
- A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved.
- Dividing INTEGERS (x / y) yields a NUMERIC result. You can use the // operator to truncate the result to a whole number.

Examples

The following are examples of number-type literals:

```
42
3.5
4.
.001
5e2
1.925e-3
```

Scientific notation :

```
=> SELECT NUMERIC '1e10';
?column?
-----
10000000000
(1 row)
```

BINARY scaling:

```
=> SELECT NUMERIC '1p10';  
?column?  
-----  
      1024  
(1 row)  
=> SELECT FLOAT 'Infinity';  
?column?  
-----  
Infinity  
(1 row)
```

The following examples illustrated using the / and // operators to divide integers:

```
=> SELECT 40/25;  
?column?  
-----  
1.6000000000000000  
(1 row)  
=> SELECT 40//25;  
?column?  
-----  
      1  
(1 row)
```

See Also

[Data Type Coercion](#)

String Literals

String literals are string values surrounded by single or double quotes. Double-quoted strings are subject to the backslash, but single-quoted strings do not require a backslash, except for `\` and `\\`.

You can embed single quotes and backslashes into single-quoted strings.

To include other backslash (escape) sequences, such as `\t` (tab), you must use the double-quoted form.

Precede single-quoted strings with a space between the string and its preceding word, since single quotes are allowed in identifiers.

See Also

- [SET STANDARD_CONFORMING_STRINGS](#)
- [SET ESCAPE_STRING_WARNING](#)
- [Internationalization Parameters](#)
- [Implement Locales for International Data Sets](#)

Character String Literals

Character string literals are a sequence of characters from a predefined character set and are enclosed by single quotes. If the single quote is part of the sequence, it must be doubled as `''`.

Syntax

`'characters'`

Parameters

<code>characters</code>	Arbitrary sequence of characters bounded by single quotes ('')
-------------------------	--

Single Quotes in a String

The SQL standard way of writing a single-quote character within a string literal is to write two adjacent single quotes. For example:

```
=> SELECT 'Chester''s gorilla';
   ?column?
-----
Chester's gorilla
(1 row)
```

Standard Conforming Strings and Escape Characters

Vertica uses standard conforming strings as specified in the SQL standard, which means that backslashes are treated as string literals, not escape characters.

Note: Earlier versions of Vertica did not use standard conforming strings, and backslashes were always considered escape sequences. To revert to this older behavior, set the `StandardConformingStrings` parameter to '0', as described in [Configuration Parameters](#) in the Administrator's Guide.

Examples

```
=> SELECT 'This is a string';
   ?column?
-----
This is a string
(1 row)
=> SELECT 'This \is a string';
      WARNING: nonstandard use of escape in a string literal at character 8
      HINT: Use the escape string syntax for escapes, e.g., E'\r\n'.
   ?column?
-----
This is a string
(1 row)
vmartdb=> SELECT E'This \is a string';
   ?column?
-----
This is a string
=> SELECT E'This is a \n new line';
   ?column?
-----
This is a
new line
(1 row)
=> SELECT 'String''s characters';
   ?column?
-----
```

```
String's characters  
(1 row)
```

See Also

- [SET STANDARD_CONFORMING_STRINGS](#)
- [SET ESCAPE_STRING_WARNING](#)
- [Internationalization Parameters](#)
- [Implement Locales for International Data Sets](#)

Dollar-Quoted String Literals

Dollar-quoted string literals are rarely used, but are provided here for your convenience.

The standard syntax for specifying string literals can be difficult to understand. To allow more readable queries in such situations, Vertica SQL provides dollar quoting. Dollar quoting is not part of the SQL standard, but it is often a more convenient way to write complicated string literals than the standard-compliant single quote syntax.

Syntax

```
$$characters$$
```

Parameters

<i>characters</i>	Arbitrary sequence of characters bounded by paired dollar signs (\$\$)
-------------------	--

Dollar-quoted string content is treated as a literal. Single quote, backslash, and dollar sign characters have no special meaning within a dollar-quoted string.

Notes

A dollar-quoted string that follows a keyword or identifier must be separated from the preceding word by whitespace; otherwise, the dollar-quoting delimiter is taken as part of the preceding identifier.

Examples

```
=> SELECT $$Fred's\n car$$;
      ?column?
-----
Fred's\n car
(1 row)

=> SELECT 'SELECT 'fact'';
ERROR: syntax error at or near "';'" at character 21
LINE 1: SELECT 'SELECT 'fact'';

=> SELECT 'SELECT $$fact'$$;
      ?column?
-----
SELECT $$fact
(1 row)

=> SELECT 'SELECT ''fact''';
      ?column?
-----
SELECT 'fact';
(1 row)
```

Unicode String Literals

Syntax

U&'characters' [UESCAPE '<Unicode escape character>']

Parameters

<i>characters</i>	Arbitrary sequence of UTF-8 characters bounded by single quotes (')
<i>Unicode escape character</i>	A single character from the source language character set other than a hexit, plus sign (+), quote ('), double quote ("), or white space

Using Standard Conforming Strings

With `StandardConformingStrings` enabled, Vertica supports SQL standard Unicode character string literals (the character set is UTF-8 only).

Before you enter a Unicode character string literal, enable standard conforming strings in one of the following ways.

- To enable for all sessions, update the `StandardConformingStrings` configuration parameter. See [Configuration Parameters](#) in the Administrator's Guide.
- To treat backslashes as escape characters for the current session, use the [SET STANDARD_CONFORMING_STRINGS](#) statement.

See also [Extended String Literals](#).

Examples

To enter a Unicode character in hexadecimal, such as the Russian phrase for "thank you, use the following syntax:

```
=> SET STANDARD_CONFORMING_STRINGS TO ON;
=> SELECT U&' \0441\043F\0430\0441\0438\0431\043E' as 'thank you';
  thank you
-----
  спасибо
(1 row)
```

To enter the German word `müde` (where `u` is really `u-umlaut`) in hexadecimal:

```
=> SELECT U&'m\00fcde';
?column?
-----
müde
(1 row)
=> SELECT 'ü';
?column?
-----
ü
(1 row)
```

To enter the `LINEAR B IDEOGRAM B240 WHEELED CHARIOT` in hexadecimal:

```
=> SELECT E'\xF0\x90\x83\x8C';
?column?
-----
(wheeled chariot character)
(1 row)
```

Note: Not all fonts support the wheeled chariot character.

See Also

- [SET STANDARD_CONFORMING_STRINGS](#)
- [SET ESCAPE_STRING_WARNING](#)
- [Internationalization Parameters](#)
- [Implement Locales for International Data Sets](#)

VARBINARY String Literals

VARBINARY string literals allow you to specify hexadecimal or binary digits in a string literal.

Syntax

```
X'<hexadecimal digits>'
B'<binary digits>'
```

Parameters

X or x	Specifies hexadecimal digits. The <hexadecimal digits> string must be enclosed in single quotes (').
B or b	Specifies binary digits. The <binary digits> string must be enclosed in single quotes (').

Examples

```
=> SELECT X'abcd';
?column?
-----
\253\315
(1 row)

=> SELECT B'101100';
?column?
-----
'
(1 row)
```

Extended String Literals

Syntax

`E'characters'`

Parameters

<code>characters</code>	Arbitrary sequence of characters bounded by single quotes ('')
-------------------------	--

You can use C-style backslash sequence in extended string literals, which are an extension to the SQL standard. You specify an extended string literal by writing the letter E as a prefix (before the opening single quote); for example:

```
E'extended character string\n'
```

Within an extended string, the backslash character (\) starts a C-style backslash sequence, in which the combination of backslash and following character or numbers represent a special byte value, as shown in the following list. Any other character following a backslash is taken literally; for example, to include a backslash character, write two backslashes (\\).

- \\ is a backslash
- \b is a backspace
- \f is a form feed
- \n is a newline
- \r is a carriage return
- \t is a tab
- \x##, where ## is a 1 or 2-digit hexadecimal number; for example \x07 is a tab
- \###, where ### is a 1, 2, or 3-digit octal number representing a byte with the corresponding code.

When an extended string literal is concatenated across lines, write only E before the first opening quote:

```
=> SELECT E'first part o'  
       'f a long line';  
       ?column?  
-----  
first part of a long line  
(1 row)
```

Two adjacent single quotes are used as one single quote:

```
=> SELECT 'Aren''t string literals fun?';  
       ?column?  
-----  
Aren't string literals fun?  
(1 row)
```

Standard Conforming Strings and Escape Characters

When interpreting commands, such as those entered in `vsq` or in queries passed via JDBC or ODBC, Vertica uses standard conforming strings as specified in the SQL standard. In standard conforming strings, backslashes are treated as string literals (ordinary characters), not escape characters.

Note: Text read in from files or streams (such as the data inserted using the [COPY](#) statement) are not treated as literal strings. The `COPY` command defines its own escape characters for the data it reads. See the [COPY](#) statement documentation for details.

The following options are available, but Micro Focus recommends that you migrate your application to use standard conforming strings at your earliest convenience, after warnings have been addressed.

- To revert to pre 4.0 behavior, set the `StandardConformingStrings` parameter to '0', as described in [Configuration Parameters](#) in the Administrator's Guide.
- To enable standard conforming strings permanently, set the `StandardConformingStrings` parameter to '1', as described in the procedure in the section, "Identifying Strings that are not Standard Conforming," below.
- To enable standard conforming strings per session, use [SET STANDARD_CONFORMING_STRING TO ON](#), which treats backslashes as escape characters for the current session only.

The two sections that follow help you identify issues between Vertica 3.5 and 4.0.

Identifying Strings That Are Not Standard Conforming

The following procedure can be used to identify nonstandard conforming strings in your application so that you can convert them into standard conforming strings:

1. Be sure the `StandardConformingStrings` parameter is off, as described in [Internationalization Parameters](#) in the Administrator's Guide.

```
=> ALTER DATABASE mydb SET StandardConformingStrings = 0;
```

Note: Micro Focus recommends that you migrate your application to use Standard Conforming Strings at your earliest convenience.

2. Turn on the `EscapeStringWarning` parameter. (ON is the default in Vertica Version 4.0 and later.)

```
=> ALTER DATABASE mydb SET EscapeStringWarning = 1;
```

Vertica now returns a warning each time it encounters an escape string within a string literal. For example, Vertica interprets the `\n` in the following example as a new line:

```
=> SELECT 'a\nb';
WARNING: nonstandard use of escape in a string literal at character 8
HINT: Use the escape string syntax for escapes, e.g., E'\r\n'.
?column?
-----
a
b
(1 row)
```

When `StandardConformingStrings` is ON, the string is interpreted as four characters: `a \ n b`.

Modify each string that Vertica flags by extending it as in the following example:

```
E'a\nb'
```

Or if the string has quoted single quotes, double them; for example, `'one'' double'`.

3. Turn on the `StandardConformingStrings` parameter for all sessions:

```
=> ALTER DATABASE mydb SET StandardConformingStrings = 1;
```

Doubled Single Quotes

This section discusses vsql inputs that are not passed on to the server.

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, the following inputs, 'You 're here!' ignored the second consecutive quote and returns the following:

```
=> SELECT 'You're here!';
      ?column?
-----
You're here!
(1 row)
```

This is the SQL standard representation and is preferred over the form, 'You\'re here!', because backslashes are not parsed as before. You need to escape the backslash:

```
=> SELECT (E'You\'re here!');
      ?column?
-----
You're here!
(1 row)
```

This behavior change introduces a potential incompatibility in the use of the vsql [\set](#) command, which automatically concatenates its arguments. For example, the following works in both Vertica 3.5 and 4.0:

```
\set file '\'' `pwd` '/file.txt' '\''\echo :file
```

vsql takes the four arguments and outputs the following:

```
'/home/vertica/file.txt'
```

In Vertica 3.5 the above `\set file` command could be written all with the arguments run together, but in 4.0 the adjacent single quotes are now parsed differently:

```
\set file '\''`pwd`'/file.txt'\'\echo :file
'/home/vertica/file.txt''
```

Note the extra single quote at the end. This is due to the pair of adjacent single quotes together with the backslash-quoted single quote.

The extra quote can be resolved either as in the first example above, or by combining the literals as follows:

```
\set file '\''`pwd`'/file.txt'\'\echo :file
'/home/vertica/file.txt'
```

In either case the backslash-quoted single quotes should be changed to doubled single quotes as follows:

```
\set file '''' `pwd` '/file.txt''''
```

Additional Examples

```
=> SELECT 'This \is a string';
      ?column?
-----
This \is a string
(1 row)

=> SELECT E'This \is a string';
      ?column?
-----
This is a string

=> SELECT E'This is a \n new line';
      ?column?
-----
This is a
new line
(1 row)

=> SELECT 'String''s characters';
      ?column?
-----
String's characters
(1 row)
```

Date/Time Literals

Date or time literal input must be enclosed in single quotes. Input is accepted in almost any reasonable format, including ISO 8601, SQL-compatible, traditional POSTGRES, and others.

Vertica handles date/time input more flexibly than the SQL standard requires. The exact parsing rules of date/time input and for the recognized text fields including months, days of the week, and time zones are described in [Date/Time Expressions](#).

Time Zone Values

Vertica attempts to be compatible with the SQL standard definitions for time zones. However, the SQL standard has an odd mix of date and time types and capabilities. Obvious problems are:

- Although the [DATE](#) type does not have an associated time zone, the [TIME/TIMETZ](#) type can. Time zones in the real world have little meaning unless associated with a date as well as a time, since the offset can vary through the year with daylight-saving time boundaries.
- Vertica assumes your local time zone for any data type containing only date or time.
- The default time zone is specified as a constant numeric offset from UTC. It is therefore not possible to adapt to daylight-saving time when doing date/time arithmetic across DST boundaries.

To address these difficulties, Micro Focus recommends using Date/Time types that contain both date and time when you use time zones. Micro Focus recommends that you do *not* use the type `TIME WITH TIME ZONE`, even though it is supported it for legacy applications and for compliance with the SQL standard.

Time zones and time-zone conventions are influenced by political decisions, not just earth geometry. Time zones around the world became somewhat standardized during the 1900's, but continue to be prone to arbitrary changes, particularly with respect to daylight-savings rules.

Vertica currently supports daylight-savings rules over the time period 1902 through 2038, corresponding to the full range of conventional UNIX system time. Times outside that range are taken to be in "standard time" for the selected time zone, no matter what part of the year in which they occur.

Example	Description
PST	Pacific Standard Time
-8:00	ISO-8601 offset for PST
-800	ISO-8601 offset for PST
-8	ISO-8601 offset for PST
zulu	Military abbreviation for UTC
z	Short form of zulu

Day of the Week Names

The following tokens are recognized as names of days of the week:

Day	Abbreviations
SUNDAY	SUN
MONDAY	MON
TUESDAY	TUE, TUES
WEDNESDAY	WED, WEDS
THURSDAY	THU, THUR, THURS
FRIDAY	FRI
SATURDAY	SAT

Month Names

The following tokens are recognized as names of months:

Month	Abbreviations
JANUARY	JAN
FEBRUARY	FEB

Month	Abbreviations
MARCH	MAR
APRIL	APR
MAY	MAY
JUNE	JUN
JULY	JUL
AUGUST	AUG
SEPTEMBER	SEP, SEPT
OCTOBER	OCT
NOVEMBER	NOV
DECEMBER	DEC

Interval Literal

A literal that represents a time span.

Syntax

```
[ @ ] [-] { quantity subtype-unit }[...] [ AGO ]
```

Parameters

@	Ignored
- (minus)	Specifies a negative interval value.
<i>quantity</i>	Integer numeric constant
<i>subtype-unit</i>	See Interval Subtype Units for valid values. Subtype units must be specified for year-month intervals; they are optional for day-time intervals.
AGO	Specifies a negative interval value. AGO and - (minus) are synonymous.

Notes

- The amounts of different units are implicitly added up with appropriate sign accounting.
- The boundaries of an interval constant are:
 - 9223372036854775807 usec to -9223372036854775807 usec
 - 296533 years 3 mons 21 days 04:00:54.775807 to -296533 years -3 mons -21 days -04:00:54.775807
- The range of an interval constant is $\pm 2^{63} - 1$ (plus or minus two to the sixty-third minus one) microseconds.
- In Vertica, interval fields are additive and accept large floating-point numbers.

Examples

See [Specifying Interval Input.Interval Literal](#)

Interval Subtype Units

The following tables lists subtype units that can be specified in an interval literal, divided into major categories:

- [Year-month subtype units](#)
- [Day-time subtype units](#)

Year-month subtype units

Subtypes	Units	Notes
Millennium	mil millennium	
	millennia mils	
Century	c cent century	
	centuries	
Decade	dec decade	
	decades decs	
Year	a	Julian year: 365.25 days
	ka	Julian kilo-year: 365250 days
	y year yr	Calendar year: 365 days
	years yrs	
Quarter	q qtr quarter	
	qtrs quarters	
Month	mon month m	Vertica can interpret m as minute or month, depending on context. See Processing m Input below.
	mons months	
Week	w week	
	weeks	

Day-time subtype units

Subtypes	Units	Notes	
Day	d day		
	days		
Hour	h hour hr		
	hours hrs		
Minute	min minute mm m		Vertica can interpret input unit m as minute or month, depending on context. See Processing m Input below.
	mins minutes		
Second	s sec second		
	seconds secs		
Millisecond	millisecond ms		
	milliseconds mseconds msec		
Microsecond	microsecond us usec		
	microseconds useconds usecs		

Processing m Input

Vertica uses context to interpret the input unit `m` as months or minutes. For example, the following command creates a one-column table with an interval value:

```
=> CREATE TABLE int_test(i INTERVAL YEAR TO MONTH);
```

Given the following `INSERT` statement, Vertica interprets the interval literal `1y 6m` as 1 year 6 months:

```
=> INSERT INTO int_test VALUES('1y 6m');
OUTPUT
-----
      1
(1 row)

=> SET INTERVALSTYLE TO UNITS;
=> SELECT * FROM int_test;
      i
-----
1 year 6 months
(1 row)
```

The following `ALTER TABLE` statement adds a `DAY TO MINUTE` interval column to table `int_test`:

```
=> ALTER TABLE int_test ADD COLUMN x INTERVAL DAY TO MINUTE;
ALTER TABLE
```

The next `INSERT` statement sets the first and second columns to `3y 20m` and `1y 6m`, respectively. In this case, Vertica interprets the `m` input literals in two ways:

- For column `i`, Vertica interprets the `m` input as months, and displays 4 years 8 months.
- For column `x`, Vertica interprets the `m` input as minutes. Because the interval is defined as `DAY TO MINUTE`, it converts the inserted input value `1y 6m` to 365 days 6 minutes:

```
=> INSERT INTO int_test VALUES ('3y 20m', '1y 6m');
OUTPUT
-----
      1
(1 row)

=> SELECT * FROM int_test;
      i          |          x
-----+-----
1 year 6 months |
4 years 8 months | 365 days 6 mins
(2 rows)
```

Interval Qualifier

Specifies how to interpret and format an [interval literal](#) for output, and, optionally, sets precision. Interval qualifiers are composed of one or two units:

```
unit[p] [ TO unit[p] ]
```

where:

- *unit* specifies a day-time or year-month [subtype](#).
- *p* specifies precision, an integer between 0 and 6. In general, precision only applies to SECOND units. The default precision for SECOND is 6. For details, see [Specifying Interval Precision](#).

If an interval omits an interval qualifier, the default is DAY TO SECOND(6).

Interval qualifiers are divided into two categories: [day-time](#) and [year-month](#), as shown in the tables below.

Day-time interval qualifiers

Qualifier	Description
DAY	Unconstrained
DAY TO HOUR	Span of days and hours
DAY TO MINUTE	Span of days and minutes
DAY TO SECOND	Span of days, hours, minutes, seconds, and fractions of a second.
HOUR	Hours within days
HOUR TO MINUTE	Span of hours and minutes
HOUR TO SECOND	Span of hours and seconds
MINUTE	Minutes within hours
MINUTE TO SECOND	Span of minutes and seconds
SECOND	Seconds within minutes

Note: The SECOND field can have an interval fractional seconds precision, which indicates the number of decimal digits maintained following the decimal point in the SECONDS

Day-time interval qualifiers, continued

Qualifier	Description
	value. When SECOND is not the first field, it has a precision of 2 places before the decimal point.

Year-month interval qualifiers

YEAR	Unconstrained
MONTH	Months within year
YEAR TO MONTH	Span of years and months

Note: Vertica also supports INTERVALYM, which is an alias for INTERVAL YEAR TO MONTH. Thus, the following two statements are equivalent:

```
=> SELECT INTERVALYM '1 2';  
?column?  
-----  
1 year 2 months  
  
=> SELECT INTERVAL '1 2' YEAR TO MONTH;  
?column?  
-----  
1 year 2 months  
(1 row)
```

Examples

See [Controlling Interval Format](#).

Operators

Operators are logical, mathematical, and equality symbols used in SQL to evaluate, compare, or calculate values.

Binary Operators

Each of the functions in the following table works with BINARY and VARBINARY data types.

Operator	Function	Description
'='	binary_eq	Equal to
'<>'	binary_ne	Not equal to
'<'	binary_lt	Less than
'<='	binary_le	Less than or equal to
'>'	binary_gt	Greater than
'>='	binary_ge	Greater than or equal to
'&'	binary_and	And
'~'	binary_not	Not
' '	binary_or	Or
'#'	binary_xor	Either or
' '	binary_cat	Concatenate

Notes

- If the arguments vary in length binary operators treat the values as though they are all equal in length by right-extending the smaller values with the zero byte to the full width of the column (except when using the `binary_cat` function). For example, given the values 'ff' and 'f', the value 'f' is treated as 'f0'.

- Operators are strict with respect to nulls. The result is null if any argument is null. For example, `null <> 'a'::binary` returns null.
- To apply the OR (`'|'`) operator to a `VARBINARY` type, explicitly cast the arguments; for example:

```
=> SELECT '1'::VARBINARY | '2'::VARBINARY;
?column?
-----
      3
(1 row)
```

Similarly, to apply the [LENGTH](#), [REPEAT](#), [TO_HEX](#), and [SUBSTRING](#) functions to a `BINARY` type, explicitly cast the argument; for example:

```
=> SELECT LENGTH('\001\002\003\004'::varbinary(4));
LENGTH
-----
      4
(1 row)
```

When applying an operator or function to a column, the operator's or function's argument type is derived from the column type.

Examples

In the following example, the zero byte is not removed from column `cat1` when values are concatenated:

```
=> SELECT 'ab'::BINARY(3) || 'cd'::BINARY(2) AS cat1,
'ab'::VARBINARY(3) ||
'cd'::VARBINARY(2) AS cat2;
cat1 | cat2
-----+-----
ab\000cd | abcd
(1 row)
```

When the binary value `'ab'::binary(3)` is translated to varbinary, the result is equivalent to `'ab\000'::varbinary(3)`; for example:

```
=> SELECT 'ab'::binary(3); binary
-----
ab\000
(1 row)
```

The following example performs a bitwise AND operation on the two input values (see also [BIT_AND](#)):

```
=> SELECT '10001' & '011' as AND;  
AND  
-----  
1  
(1 row)
```

The following example performs a bitwise OR operation on the two input values (see also [BIT_OR](#)):

```
=> SELECT '10001' | '011' as OR;  
OR  
-----  
10011  
(1 row)
```

The following example concatenates the two input values:

```
=> SELECT '10001' || '011' as CAT;  
CAT  
-----  
10001011  
(1 row)
```

Boolean Operators

Syntax

[AND | OR | NOT]

Parameters

SQL uses a three-valued Boolean logic where the null value represents "unknown."

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL

a	b	a AND b	a OR b
NULL	NULL	NULL	NULL

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Notes

- The operators AND and OR are commutative, that is, you can switch the left and right operand without affecting the result. However, the order of evaluation of subexpressions is not defined. When it is essential to force evaluation order, use a [CASE](#) construct.
- Do not confuse Boolean operators with the [Boolean-Predicate](#) or the [Boolean](#) data type, which can have only two values: true and false.

Comparison Operators

Comparison operators are available for all data types where comparison makes sense. All comparison operators are binary operators that return values of True, False, or NULL.

Syntax and Parameters

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
= or <=>	equal
<> or !=	not equal

Notes

- The comparison operators return NULL (signifying "unknown") when either operand is null.
- The <=> operator performs an equality comparison like the = operator, but it returns true, instead of NULL, if both operands are NULL, and false, instead of NULL, if one operand is NULL.

Data Type Coercion Operators (CAST)

Data type coercion (casting) passes an expression value to an input conversion routine for a specified data type, resulting in a constant of the indicated type.

Syntax

```
SELECT CAST ( expression AS data_type )
```

```
SELECT expression::data_type
```

```
SELECT data_type 'string'
```

Parameters

<i>expression</i>	An expression of any type
<i>data_type</i>	Converts the value of <i>expression</i> to one of the following data types: <ul style="list-style-type: none">• BINARY• BOOLEAN• CHARACTER• DATE/TIME• NUMERIC

Notes

- In Vertica, data type coercion (casting) can be invoked by an explicit cast request. It must use one of the following constructs:

```
=> SELECT CAST ( expression AS data_type )  
=> SELECT expression::data_type  
=> SELECT data_type 'string'
```

- The explicit type cast can be omitted if there is no ambiguity as to the type the constant must be. For example, when a constant is assigned directly to a column, it is automatically coerced to the column's data type.
- If a binary value is cast (implicitly or explicitly) to a binary type with a smaller length, the value is silently truncated. For example:

```
=> SELECT 'abcd'::BINARY(2);  
?column?  
-----  
ab  
(1 row)
```

- Similarly, if a character value is cast (implicitly or explicitly) to a character value with a smaller length, the value is silently truncated. For example:

```
=> SELECT 'abcd'::CHAR(3);  
?column?  
-----  
abc
```

- Vertica supports only casts and resize operations as follows:
 - BINARY to and from VARBINARY
 - VARBINARY to and from LONG VARBINARY
 - BINARY to and from LONG VARBINARY
- On binary data that contains a value with fewer bytes than the target column, values are right-extended with the zero byte '\0' to the full width of the column. Trailing zeros on variable-length binary values are not right-extended:

```
=> SELECT 'ab'::BINARY(4), 'ab'::VARBINARY(4), 'ab'::LONG VARBINARY(4);
?column? | ?column? | ?column?
-----+-----+-----
ab\000\000 | ab      | ab
(1 row)
```

Examples

```
=> SELECT CAST((2 + 2) AS VARCHAR);
?column?
-----
4
(1 row)

=> SELECT (2 + 2)::VARCHAR;
?column?
-----
4
(1 row)

=> SELECT INTEGER '123';
?column?
-----
123
(1 row)

=> SELECT (2 + 2)::LONG VARCHAR
?column?
-----
4
(1 row)

=> SELECT '2.2' + 2;
ERROR:  invalid input syntax for integer: "2.2"

=> SELECT FLOAT '2.2' + 2;
?column?
-----
4.2
(1 row)
```

See Also

- [Data Type Conversions](#)
- [Data Type Coercion Chart](#)
- [CAST Failures](#)

Cast Failures

When you invoke data type coercion (casting) by an explicit cast and the cast fails, the result returns either an error or NULL. Cast failures commonly occur when you attempt to cast conflicting conversions, such as trying to convert a varchar expression that contains letters to an int.

When a cast fails, the result returned depends on the data type.

Data Type	Cast Failure Default
Date/Time	NULL
Literal	Error
All Other Types	Error

Enabling Strict Time Casts

You can enable all cast failures to result in an error, including those for Date/Time data types. Doing so allows you to see the reason why some or all of the cast failed. To return an error instead of NULL, use the [ALTER SESSION](#) statement with the SET parameter:

```
ALTER SESSION SET EnableStrictTimeCasts=1;
```

The following example shows a Date/Time cast failure that returns NULL:

```
=> CREATE TABLE mytable (a VARCHAR);
CREATE TABLE
=> INSERT INTO mytable VALUES('string');
OUTPUT
-----
1
(1 row)
=> INSERT INTO mytable VALUES('1');
OUTPUT
-----
1
(1 row)
=> SELECT a::time FROM mytable;
a
---
(2 rows)
```

When you specify `EnableStrictTimeCasts`, the cast failure returns an error:

```
=> ALTER SESSION SET EnableStrictTimeCasts=1;
ALTER SESSION
=> SELECT a::time FROM mytable;
ERROR 2005: Invalid input syntax for time: "1"
```

Returning All Cast Failures as NULL

To explicitly cast an expression to a requested data type, use the following construct:

```
SELECT expression::data_type
```

Using this command to cast any values to a conflicting data type returns the following error:

```
ERROR 2827: Could not convert "string" from column table.a to an int8
```

In addition to the `::` cast, Vertica supports the use of `::!`. Use `::!` instead of `::`, *if you want to return:*

- NULL instead of an error for any non-Date/Time data types
- NULL instead of an error after setting `EnableStrictTimeCasts`

Returning all cast failures as NULL allows those expressions that succeeded during the cast to appear in the result. Those expressions which failed during the cast, however, have a NULL value.

The following example shows a cast failure that returns an error:

```
=> CREATE TABLE mytable (a VARCHAR);
CREATE TABLE
=> INSERT INTO mytable VALUES('string');
OUTPUT
-----
1
(1 row)
=> INSERT INTO mytable VALUES('1');
OUTPUT
-----
1
(1 row)
=> SELECT a::int FROM mytable;
ERROR 2827: Could not convert "string" from column mytable.a to an int8
```

When you use `::!`, the cast fails for the "string" value and returns NULL. However, it succeeds for the "1" value and returns 1:

```
=> SELECT a::!int FROM mytable;
a
---
1
```

(2 rows)

Date/Time Operators

Syntax

[+ | - | * | /]

Parameters

+ Addition
- Subtraction
* Multiplication
/ Division

Notes

- The operators described below that take TIME or TIMESTAMP inputs actually come in two variants: one that takes TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE, and one that takes TIME WITHOUT TIME ZONE or TIMESTAMP WITHOUT TIME ZONE. For brevity, these variants are not shown separately.
- The + and * operators come in commutative pairs (for example both DATE + INTEGER and INTEGER + DATE); only one of each such pair is shown.

Example	Result Type	Result
DATE '2001-09-28' + INTEGER '7'	DATE	'2001-10-05'
DATE '2001-09-28' + INTERVAL '1 HOUR'	TIMESTAMP	'2001-09-28 01:00:00'
DATE '2001-09-28' + TIME '03:00'	TIMESTAMP	'2001-09-28 03:00:00'
INTERVAL '1 DAY' + INTERVAL '1 HOUR'	INTERVAL	'1 DAY 01:00:00'
TIMESTAMP '2001-09-28 01:00' + INTERVAL '23 HOURS'	TIMESTAMP	'2001-09-29 00:00:00'
TIME '01:00' + INTERVAL '3 HOURS'	TIME	'04:00:00'

Example	Result Type	Result
- INTERVAL '23 HOURS'	INTERVAL	'-23:00:00'
DATE '2001-10-01' - DATE '2001-09-28'	INTEGER	'3'
DATE '2001-10-01' - INTEGER '7'	DATE	'2001-09-24'
DATE '2001-09-28' - INTERVAL '1 HOUR'	TIMESTAMP	'2001-09-27 23:00:00'
TIME '05:00' - TIME '03:00'	INTERVAL	'02:00:00'
TIME '05:00' INTERVAL '2 HOURS'	TIME	'03:00:00'
TIMESTAMP '2001-09-28 23:00' - INTERVAL '23 HOURS'	TIMESTAMP	'2001-09-28 00:00:00'
INTERVAL '1 DAY' - INTERVAL '1 HOUR'	INTERVAL	'1 DAY -01:00:00'
TIMESTAMP '2001-09-29 03:00' - TIMESTAMP '2001-09-27 12:00'	INTERVAL	'1 DAY 15:00:00'
900 * INTERVAL '1 SECOND'	INTERVAL	'00:15:00'
21 * INTERVAL '1 DAY'	INTERVAL	'21 DAYS'
DOUBLE PRECISION '3.5' * INTERVAL '1 HOUR'	INTERVAL	'03:30:00'
INTERVAL '1 HOUR' / DOUBLE PRECISION '1.5'	INTERVAL	'00:40:00'

Mathematical Operators

Mathematical operators are provided for many data types.

Operator	Description	Example	Result
!	Factorial	5 !	120
+	Addition	2 + 3	5
-	Subtraction	2 - 3	-1
*	Multiplication	2 * 3	6
/	Division (integer division produces NUMERIC results).	4 / 2	2.00...

Operator	Description	Example	Result
//	With integer division, returns an INTEGER rather than a NUMERIC.	117.32 // 2.5	46
%	Modulo (remainder)	5 % 4	1
^	Exponentiation	2.0 ^ 3.0	8
/	Square root	/ 25.0	5
/	Cube root	/ 27.0	3
!!	Factorial (prefix operator)	!! 5	120
@	Absolute value	@ -5.0	5
&	Bitwise AND	91 & 15	11
	Bitwise OR	32 3	35
#	Bitwise XOR	17 # 5	20
~	Bitwise NOT	~1	-2
<<	Bitwise shift left	1 << 4	16
>>	Bitwise shift right	8 >> 2	2

Notes

- The bitwise operators work only on integer data types, whereas the others are available for all numeric data types.
- Vertica supports the use of the factorial operators on positive and negative floating point (DOUBLE PRECISION) numbers as well as integers. For example:

```
=> SELECT 4.98!;
   ?column?
-----
 115.978600750905
(1 row)
```

- Factorial is defined in term of the gamma function, where $(-1) = \text{Infinity}$ and the other negative integers are undefined. For example:

$$(-4)! = \text{NaN}$$

$$-(4!) = -24$$

- Factorial is defined as $z! = \text{gamma}(z+1)$ for all complex numbers z . See the [Handbook of Mathematical Functions](#) (1964) Section 6.1.5.
- See `MOD()` for details about the behavior of `%`.

NULL Operators

To check whether a value is or is not NULL, use the constructs:

```
[expression IS NULL | expression IS NOT NULL]
```

Alternatively, use equivalent, but nonstandard, constructs:

```
[expression ISNULL | expression NOTNULL]
```

Do not write `expression = NULL` because NULL represents an unknown value, and two unknown values are not necessarily equal. This behavior conforms to the SQL standard.

Note: Some applications might expect that `expression = NULL` returns true if `expression` evaluates to null. Vertica strongly recommends that these applications be modified to comply with the SQL standard.

String Concatenation Operators

To concatenate two strings on a single line, use the concatenation operator (two consecutive vertical bars).

Syntax

```
string || string
```

Parameters

<i>string</i>	Is an expression of type CHAR or VARCHAR
---------------	--

Notes

- `||` is used to concatenate expressions and constants. The expressions are cast to VARCHAR if possible, otherwise to VARBINARY, and must both be one or the other.
- Two consecutive strings within a single SQL statement on separate lines are automatically concatenated

Examples

The following example is a single string written on two lines:

```
=> SELECT E'xx'-> '\\';  
?column?  
-----  
xx\  
(1 row)
```

The following examples show two strings concatenated:

```
=> SELECT E'xx' ||-> '\\';  
?column?  
-----  
xx\  
(1 row)  
  
=> SELECT 'auto' || 'mobile';  
?column?  
-----  
automobile  
(1 row)  
  
=> SELECT 'auto'-> 'mobile';  
?column?  
-----  
automobile  
(1 row)  
  
=> SELECT 1 || 2;  
?column?  
-----  
12  
(1 row)
```

```
=> SELECT '1' || '2';  
?column?  
-----  
12  
(1 row)  
  
=> SELECT '1'-> '2';  
?column?  
-----  
12  
(1 row)
```

Expressions

SQL expressions are the components of a query that compare a value or values against other values. They can also perform calculations. Expressions found inside any SQL command are usually in the form of a conditional statement.

Operator Precedence

The following table shows operator precedence in decreasing (high to low) order.

Note: When an expression includes more than one operator, Micro Focus recommends that you specify the order of operation using parentheses, rather than relying on operator precedence.

Operator/Element	Associativity	Description
.	left	table/column name separator
::	left	typecast
[]	left	array element selection
-	right	unary minus
^	left	exponentiation
* / %	left	multiplication, division, modulo
+ -	left	addition, subtraction
IS		IS TRUE, IS FALSE, IS UNKNOWN, IS NULL
IN		set membership
BETWEEN		range containment
OVERLAPS		time interval overlap
LIKE		string pattern matching
< >		less than, greater than

Operator/Element	Associativity	Description
=	right	equality, assignment
NOT	right	logical negation
AND	left	logical conjunction
OR	left	logical disjunction

Expression Evaluation Rules

The order of evaluation of subexpressions is not defined. In particular, the inputs of an operator or function are not necessarily evaluated left-to-right or in any other fixed order. To force evaluation in a specific order, use a CASE construct. For example, this is an untrustworthy way of trying to avoid division by zero in a WHERE clause:

```
=> SELECT x, y WHERE x <> 0 AND y/x > 1.5;
```

But this is safe:

```
=> SELECT x, y
WHERE
  CASE
    WHEN x <> 0 THEN y/x > 1.5
    ELSE false
  END;
```

A CASE construct used in this fashion defeats optimization attempts, so use it only when necessary. (In this particular example, it would be best to avoid the issue by writing $y > 1.5 * x$ instead.)

Limits to SQL Expressions

There are some limits on the number of modifiers and recursions that you can make in an expression. There are two limits that you should be aware of:

- The first limit is based on the stack available to the expression. VERTICA requires at least 100kb of free stack. If this limit is exceeded then the error "The query contains an expression that is too complex to analyze" may be thrown. Adding additional physical memory and/or increasing the value of `ulimit -s` max increase the available stack and prevent the error.

- The second limit is the number of recursions possible in an analytic expression. The limit is 2000. If this limit is exceeded then the error "The query contains an expression that is too complex to analyze" may be thrown. This limit cannot be increased.

Aggregate Expressions

An aggregate expression applies an aggregate function across the rows or groups of rows selected by a query.

An aggregate expression only can appear in the select list or HAVING clause of a SELECT statement. It is invalid in other clauses such as WHERE, because those clauses are evaluated before the results of aggregates are formed.

Syntax

An aggregate expression has the following format:

```
aggregate-function ( [ * ] [ ALL | DISTINCT ] expression )
```

Parameters

<i>aggregate-function</i>	A Vertica function that aggregates data over groups of rows from a query result set.
ALL DISTINCT	Specifies which input rows to process: <ul style="list-style-type: none">• ALL (default): Invokes <i>aggregate-function</i> across all input rows where <i>expression</i> evaluates to a non-null value.• DISTINCT: Invokes <i>aggregate-function</i> across all input rows where <i>expression</i> evaluates to a unique non-null value.
<i>expression</i>	A value expression that does not itself contain an aggregate expression.

CASE Expressions

The CASE expression is a generic conditional expression that can be used wherever an expression is valid. It is similar to case and if/then/else statements in other languages.

Syntax (form 1)

```
CASE
  WHEN condition THEN result
  [ WHEN condition THEN result ]
  ...
  [ ELSE result ]
END
```

Parameters

<i>condition</i>	Is an expression that returns a boolean (true/false) result. If the result is false, subsequent WHEN clauses are evaluated in the same manner.
<i>result</i>	Specifies the value to return when the associated <i>condition</i> is true.
ELSE <i>result</i>	If no <i>condition</i> is true then the value of the CASE expression is the result in the ELSE clause. If the ELSE clause is omitted and no condition matches, the result is null.

Syntax (form 2)

```
CASE expression
  WHEN value THEN result
  [ WHEN value THEN result ]
  ...
  [ ELSE result ]
END
```

Parameters

<i>expression</i>	An expression that is evaluated and compared to all the <i>value</i> specifications in the WHEN clauses until one is found that is equal.
<i>value</i>	Specifies a value to compare to the <i>expression</i> .

<i>result</i>	Specifies the value to return when the <i>expression</i> is equal to the specified <i>value</i> .
ELSE <i>result</i>	Specifies the value to return when the <i>expression</i> is not equal to any <i>value</i> ; if no ELSE clause is specified, the value returned is null.

Notes

The data types of all the result expressions must be convertible to a single output type.

Examples

The following examples show two uses of the CASE statement.

```
=> SELECT * FROM test;
 a
 ---
 1
 2
 3

=> SELECT a,
        CASE WHEN a=1 THEN 'one'
             WHEN a=2 THEN 'two'
             ELSE 'other'
        END
FROM test;
 a | case
 ---+-----
 1 | one
 2 | two
 3 | other

=> SELECT a,
        CASE a WHEN 1 THEN 'one'
             WHEN 2 THEN 'two'
             ELSE 'other'
        END
FROM test;
 a | case
 ---+-----
 1 | one
 2 | two
 3 | other
```

Special Example

A CASE expression does not evaluate subexpressions that are not needed to determine the result. You can use this behavior to avoid division-by-zero errors:

```
=> SELECT x FROM T1 WHERE
      CASE WHEN x <> 0 THEN y/x > 1.5
      ELSE false
      END;
```

Column References

Syntax

`[schema.]table-name].column-name`

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	<p>One of the following:</p> <ul style="list-style-type: none">• The name of a table• A table alias defined in the query's FROM clause
<i>column-name</i>	<p>A column name that is unique among all queried tables.</p>

Restrictions

A column reference cannot contain any spaces.

Comments

A comment is an arbitrary sequence of characters beginning with two consecutive hyphen characters and extending to the end of the line. For example:

```
-- This is a standard SQL comment
```

A comment is removed from the input stream before further syntax analysis and is effectively replaced by white space.

Alternatively, C-style block comments can be used where the comment begins with `/*` and extends to the matching occurrence of `*/`.

```
/* multiline comment
 * with nesting: /* nested block comment */
 */
```

These block comments nest, as specified in the SQL standard. Unlike C, you can comment out larger blocks of code that might contain existing block comments.

Date/Time Expressions

Vertica uses an internal heuristic parser for all date/time input support. Dates and times are input as strings, and are broken up into distinct fields with a preliminary determination of what kind of information might be in the field. Each field is interpreted and either assigned a numeric value, ignored, or rejected. The parser contains internal lookup tables for all textual fields, including months, days of the week, and time zones.

Vertica parses date/time type inputs as follows:

1. Break the input string into tokens and categorize each token as a string, time, time zone, or number.
2. Numeric token contains:
 - colon (:) — Parse as a time string, include all subsequent digits and colons.
 - dash (-), slash (/), or two or more dots (.) — Parse as a date string which might have a text month.
 - Numeric only — Parse as a single field or an ISO 8601 concatenated date (19990113 for January 13, 1999) or time (141516 for 14:15:16).
3. Token starts with a plus (+) or minus (-): Parse as a time zone or a special field.
4. Token is a text string: match up with possible strings.
 - Perform a binary-search table lookup for the token as either a special string (for example, today), day (for example, Thursday), month (for example, January), or noise word (for example, at, on).
 - Set field values and bit mask for fields. For example, set year, month, day for today, and additionally hour, minute, second for now.

- If not found, do a similar binary-search table lookup to match the token with a time zone.
 - If still not found, throw an error.
5. Token is a number or number field:
- If eight or six digits, and if no other date fields were previously read, interpret as a "concatenated date" (19990118 or 990118). The interpretation is YYYYMMDD or YYMMDD.
 - If token is three digits and a year was already read, interpret as day of year.
 - If four or six digits and a year was already read, interpret as a time (HHMM or HHMMSS).
 - If three or more digits and no date fields were found yet, interpret as a year (this forces yy-mm-dd ordering of the remaining date fields).
 - Otherwise the date field ordering is assumed to follow the `DateStyle` setting: mm-dd-yy, dd-mm-yy, or yy-mm-dd. Throw an error if a month or day field is found to be out of range.
6. If BC is specified: negate the year and add one for internal storage. (In the Vertica implementation, 1 BC = year zero.)
7. If BC is not specified, and year field is two digits in length: adjust the year to four digits. If field is less than 70, add 2000, otherwise add 1900.

Tip: Gregorian years AD 1–99 can be entered as 4 digits with leading zeros— for example, 0099 = AD 99.

Month Day Year Ordering

For some formats, ordering of month, day, and year in date input is ambiguous and there is support for specifying the expected ordering of these fields.

Special Date/Time Values

Vertica supports several special date/time values for convenience, as shown below. All of these values need to be written in single quotes when used as constants in SQL statements.

The values `INFINITY` and `-INFINITY` are specially represented inside the system and are displayed the same way. The others are simply notational shorthands that are converted to

ordinary date/time values when read. (In particular, NOW and related strings are converted to a specific time value as soon as they are read.)

String	Valid Data Types	Description
epoch	DATE, TIMESTAMP	1970-01-01 00:00:00+00 (UNIX SYSTEM TIME ZERO)
INFINITY	TIMESTAMP	Later than all other time stamps
-INFINITY	TIMESTAMP	Earlier than all other time stamps
NOW	DATE, TIME, TIMESTAMP	Current transaction's start time Note: NOW is not the same as the NOW function.
TODAY	DATE, TIMESTAMP	Midnight today
TOMORROW	DATE, TIMESTAMP	Midnight tomorrow
YESTERDAY	DATE, TIMESTAMP	Midnight yesterday
ALLBALLS	TIME	00:00:00.00 UTC

The following SQL-compatible functions can also be used to obtain the current time value for the corresponding data type:

- [CURRENT_DATE](#)
- [CURRENT_TIME](#)
- [CURRENT_TIMESTAMP](#)
- [LOCALTIME](#)
- [LOCALTIMESTAMP](#)

The latter four accept an optional precision specification. (See [Date/Time Functions](#).) However, these functions are SQL functions and are not recognized as data input strings.

NULL Value

NULL is a reserved keyword used to indicate that a data value is unknown.

Be very careful when using NULL in expressions. NULL is not greater than, less than, equal to, or not equal to any other expression. Use the [Boolean-Predicate](#) for determining whether an expression value is NULL.

Notes

- Vertica stores data in projections, which are sorted in a specific way. All columns are stored in ASC (ascending) order. For columns of data type NUMERIC, INTEGER, DATE, TIME, TIMESTAMP, and INTERVAL, NULL values are placed at the beginning of sorted projections (NULLS FIRST), while for columns of data type FLOAT, STRING, and BOOLEAN, NULL values are placed at the end (NULLS LAST). For details, see [NULL Sort Order](#) and [Runtime Sorting of NULL Values in Analytic Functions](#) in Analyzing Data.
- Vertica also accepts NULL characters ('\0') in constant strings and no longer removes null characters from VARCHAR fields on input or output. NULL is the ASCII abbreviation for the NULL character.
- You can write queries with expressions that contain the <=> operator for NULL=NULL joins. See [Equi-Joins and Non Equi-Joins](#) in Analyzing Data.

See Also

[NULL-handling Functions](#)

Predicates

Predicates are truth-tests. If the predicate test is true, it returns a value. Each predicate is evaluated per row, so that when the predicate is part of an entire table `SELECT` statement, the statement can return multiple results.

Predicates consist of a set of parameters and arguments. For example, in the following example `WHERE` clause:

```
WHERE name = 'Smith';
```

- `name = 'Smith'` is the predicate
- `'Smith'` is an expression

BETWEEN-predicate

The special `BETWEEN` predicate is available as a convenience.

Syntax

```
WHERE a BETWEEN x AND y
```

Examples

```
WHERE a BETWEEN x AND y
```

is equivalent to:

```
WHERE a >= x AND a <= y
```

Similarly:

```
WHERE a NOT BETWEEN x AND y
```

is equivalent to:

```
WHERE a < x OR a > y
```

You can use the `BETWEEN` predicate for date ranges:

```
=> CREATE TABLE t1 (c1 INT, c2 INT, c3 DATE);
=> COPY t1 FROM stdin DELIMITER '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1 | 2 | 2014-07-26
>> 2 | 3 | 2014-07-27
>> 3 | 4 | 2014-07-28
>> 4 | 5 | 2014-07-29
>> 5 | 6 | 2014-07-30
>> 6 | 7 | 2014-07-31
>> 7 | 8 | 2014-08-01
>> 8 | 9 | 2014-08-02
>> \.

=> SELECT* FROM t1 WHERE c3 BETWEEN DATE('2014-07-26') AND DATE('2014-07-30');
c1 | c2 | c3
-----+-----
 1 |  2 | 2014-07-26
 2 |  3 | 2014-07-27
 3 |  4 | 2014-07-28
 4 |  5 | 2014-07-29
 5 |  6 | 2014-07-30
(5 rows)
```

You can also use the NOW and INTERVAL keywords to select from a date range:

```
=> SELECT * FROM t1 WHERE c3 BETWEEN NOW()-INTERVAL '1 week' AND NOW();
c1 | c2 | c3
-----+-----
 7 |  8 | 2014-08-01
 1 |  2 | 2014-07-26
 2 |  3 | 2014-07-27
 3 |  4 | 2014-07-28
 4 |  5 | 2014-07-29
 5 |  6 | 2014-07-30
 6 |  7 | 2014-07-31
(7 rows)
```

Boolean-Predicate

Retrieves rows where the value of an expression is true, false, or unknown (null).

Syntax

```
expression IS [NOT] TRUE
expression IS [NOT] FALSE
expression IS [NOT] UNKNOWN
```

Notes

- A null input is treated as the value UNKNOWN.
- IS UNKNOWN and IS NOT UNKNOWN are effectively the same as the [NULL-predicate](#), except that the input expression does not have to be a single column value. To check a single column value for NULL, use the NULL-predicate.
- Do not confuse the boolean-predicate with [Boolean Operators](#) or the [Boolean](#) data type, which can have only two values: true and false.

Column-Value-Predicate

Syntax

column-name comparison-op constant-expression

Parameters

<i>column-name</i>	A single column of one the tables specified in the FROM Clause .
<i>comparison-op</i>	A Comparison Operators .
<i>constant-expression</i>	A constant value of the same data type as the <i>column-name</i> .

Notes

To check a column value for NULL, use the [NULL-predicate](#).

Examples

```
table.column1 = 2  
table.column2 = 'Seafood'  
table.column3 IS NULL
```

IN-predicate

Syntax

```
(column-list) [ NOT ] IN ( values-list )
```

Parameters

<i>column-list</i>	One or more comma-delimited columns in the queried tables.
<i>values-list</i>	Comma-delimited list of constant values to find in the <i>column-list</i> columns. Each <i>values-list</i> value maps to a <i>column-list</i> column according to their order in <i>values-list</i> and <i>column-list</i> , respectively. Column/value pairs must have compatible data types. You can specify multiple sets of values as follows: ((<i>values-list</i>), (<i>values-list</i>)[,...])

Examples

The following SELECT statement queries all data in table `t11`.

```
=> SELECT * FROM t11 ORDER BY pk;
pk | col1 | col2 | SKIP_ME_FLAG
-----+-----+-----+-----
 1 |  2 |  3 | t
 2 |  3 |  4 | t
 3 |  4 |  5 | f
 4 |  5 |  6 | f
 5 |  6 |  7 | t
 6 |    |  8 | f
 7 |  8 |    | t
(7 rows)
```

The following query specifies an IN predicate, to find all rows in `t11` where columns `col1` and `col2` contain values of `(2,3)` or `(6,7)`:

```
=> SELECT * FROM t11 WHERE (col1, col2) IN ((2,3), (6,7)) ORDER BY pk;
pk | col1 | col2 | SKIP_ME_FLAG
```

```
-----+-----+-----+-----  
 1 | 2 | 3 | t  
 5 | 6 | 7 | t  
(2 rows)
```

INTERPOLATE

Used to join two event series together using some ordered attribute, event series joins let you compare values from two series directly, rather than having to normalize the series to the same measurement interval.

Syntax

```
expression1 INTERPOLATE PREVIOUS VALUE expression2
```

Parameters

<i>expression1</i> <i>expression2</i>	A column reference from one the tables specified in the FROM Clause . The referenced columns are typically a DATE/TIME data type, often TIMESTAMP, inasmuch as you are joining data that represents an event series; however, the referenced columns can be of any type.
PREVIOUS VALUE	Pads the non-preserved side with the previous values from relation when there is no match. Input rows are sorted in ascending logical order of the join column. Note: An ORDER BY clause, if used, does not determine the input order but only determines query output order.

Description

- An event series join is an extension of a regular [outer join](#). Instead of padding the non-preserved side with null values when there is no match, the event series join pads the non-preserved side with the previous values from the table.

- The difference between expressing a regular outer join and an event series join is the INTERPOLATE predicate, which is used in the ON clause. See the **Examples** section below Notes and Restrictions. See also [Event Series Joins](#) in Analyzing Data.
- Data is logically partitioned on the table in which it resides, based on other ON clause equality predicates.
- Interpolated values come from the table that contains the null, not from the other table.
- Vertica does not guarantee that there will be no null values in the output. If there is no previous value for a mismatched row, that row will be padded with nulls.
- Event series join requires that both tables be sorted on columns in equality predicates, in any order, followed by the INTERPOLATED column. If data is already sorted in this order, then an explicit sort is avoided, which can improve query performance. For example, given the following tables:

```
ask: exchange, stock, ts, pricebid: exchange,  
stock, ts, price
```

In the query that follows

- ask is sorted on exchange, stock (or the reverse), ts
- bid is sorted on exchange, stock (or the reverse), ts

```
SELECT ask.price - bid.price, ask.ts, ask.stock, ask.exchange  
FROM ask FULL OUTER JOIN bid  
ON ask.stock = bid.stock AND ask.exchange =  
bid.exchange AND ask.ts INTERPOLATE PREVIOUS  
VALUE bid.ts;
```

Restrictions

- Only one INTERPOLATE expression is allowed per join.
- INTERPOLATE expressions are used only with ANSI SQL-99 syntax (the ON clause), which is already true for full outer joins.
- INTERPOLATE can be used with equality predicates only.
- The AND operator is supported but not the OR and NOT operators.
- Expressions and implicit or explicit casts are not supported, but subqueries are allowed.

Example

The examples that follow use this simple schema.

```
CREATE TABLE t(x TIME);
CREATE TABLE t1(y TIME);
INSERT INTO t VALUES('12:40:23');
INSERT INTO t VALUES('14:40:25');
INSERT INTO t VALUES('14:45:00');
INSERT INTO t VALUES('14:49:55');
INSERT INTO t1 VALUES('12:40:23');
INSERT INTO t1 VALUES('14:00:00');
COMMIT;
```

Normal Full Outer Join

```
=> SELECT * FROM t FULL OUTER JOIN t1 ON t.x = t1.y;
```

Notice the null rows from the non-preserved table:

x		y
12:40:23		12:40:23
14:40:25		
14:45:00		
14:49:55		
		14:00:00

(5 rows)

Full Outer Join with Interpolation

```
=> SELECT * FROM t FULL OUTER JOIN t1 ON t.x INTERPOLATE
PREVIOUS VALUE t1.y;
```

In this case, the rows with no entry point are padded with values from the previous row.

x		y
12:40:23		12:40:23
12:40:23		14:00:00
14:40:25		14:00:00
14:45:00		14:00:00
14:49:55		14:00:00

(5 rows)

Normal Left Outer Join

```
=> SELECT * FROM t LEFT OUTER JOIN t1 ON t.x = t1.y;
```

Again, there are nulls in the non-preserved table

x	y
12:40:23	12:40:23
14:40:25	
14:45:00	
14:49:55	

(4 rows)

Left Outer Join with Interpolation

```
=> SELECT * FROM t LEFT OUTER JOIN t1 ON t.x INTERPOLATE  
PREVIOUS VALUE t1.y;
```

Nulls padded with interpolated values.

x	y
12:40:23	12:40:23
14:40:25	14:00:00
14:45:00	14:00:00
14:49:55	14:00:00

(4 rows)

Inner Joins

For inner joins, there is no difference between a regular inner join and an event series inner join. Since null values are eliminated from the result set, there is nothing to interpolate.

A regular inner join returns only the single matching row at 12:40:23:

```
=> SELECT * FROM t INNER JOIN t1 ON t.x = t1.y;  
x      | y  
-----+-----  
12:40:23 | 12:40:23  
(1 row)
```

An event series inner join finds the same single-matching row at 12:40:23:

```
=> SELECT * FROM t INNER JOIN t1 ON t.x INTERPOLATE  
PREVIOUS VALUE t1.y;  
      x      |      y  
-----+-----  
12:40:23 | 12:40:23  
(1 row)
```

Semantics

When you write an event series join in place of normal join, values are evaluated as follows (using the schema in the above examples):

- `t` is the outer, preserved table
- `t1` is the inner, non-preserved table
- For each row in outer table `t`, the ON clause predicates are evaluated for each combination of each row in the inner table `t1`.
- If the ON clause predicates evaluate to true for any combination of rows, those combination rows are produced at the output.
- If the ON clause is false for all combinations, a single output row is produced with the values of the row from `t` along with the columns of `t1` chosen from the row in `t1` with the greatest `t1.y` value such that `t1.y < t.x`; If no such row is found, pad with nulls.

Note: `t LEFT OUTER JOIN t1` is equivalent to `t1 RIGHT OUTER JOIN t`.

In the case of a full outer join, all values from both tables are preserved.

See Also

- [Event Series Joins](#)

Join-Predicate

Specifies the columns on which records from two or more tables are joined. You can connect multiple join predicates with logical operators AND, OR, and NOT.

Syntax

`ON column-ref = column-ref [{AND | OR | NOT } column-ref = column-ref]...`

Parameters

<i>column-ref</i>	Specifies a column in a queried table. For best performance, do not join on LONG VARBINARY and LONG VARCHAR columns.
-------------------	--

LIKE-predicate

Retrieves rows where the string value of a column matches a specified pattern. The pattern can contain one or more wildcard characters.

Syntax

`string-expression [NOT] { LIKE | ILIKE | LIKEB | ILIKEB } 'pattern' [ESCAPE 'escape-character']`

Parameters

<i>string-expression</i>	The column values to search for <i>pattern</i> .
NOT	Returns true if LIKE returns false, and the reverse; equivalent to NOT <i>string</i> LIKE <i>pattern</i> .
<i>pattern</i>	Specifies what strings to match, typically containing one or both of the following wildcard characters: <ul style="list-style-type: none"> • Underscore (<code>_</code>) matches any single character. • Percent sign (<code>%</code>) matches any string of zero or more characters.
ESCAPE <i>escape-character</i>	Specifies an escape character, used in the to escape reserved characters underscore (<code>_</code>), percent (<code>%</code>), and the escape character itself. This is enforced only for non-default collations. If you omit this parameter, you can use Vertica's default escape

character, backslash (\), which is valid for CHAR and VARCHAR strings.

Note: Backslash is not valid for binary data types character. To embed an escape character for binary data types, use ESCAPE to specify a valid binary character.

Substitute Symbols

You can substitute the following symbols for LIKE and its variants:

~#	LIKEB
~~*	ILIKE
~#*	ILIKEB
!~~	NOT LIKE
!~#	NOT LIKEB
!~~*	NOT ILIKE
!~#*	NOT ILIKEB

Note: ESCAPE is not valid for the above symbols.

Pattern Matching

LIKE requires that the entire string expression match the pattern. To match a sequence of characters anywhere within a string, the pattern must start and end with a percent sign.

LIKE does not ignore trailing white space characters. If the data values to match end with an indeterminate amount of white space, append the wildcard character % to *pattern*.

LIKE Variants Compared

The LIKE predicate is compliant with the SQL standard. Vertica also supports several non-standard variants, notably ILIKE, which is equivalent to LIKE except it performs case-insensitive searches. The following differences pertain to LIKE and its variants:

- LIKE operates on UTF-8 character strings. Exact behavior depends on collation parameters such as strength. In particular, ILIKE works by setting S=2 (ignore case) in the current session locale.
- LIKE and ILIKE are stable for character strings, but immutable for binary strings, while LIKEB and ILIKEB are immutable for both cases.
- LIKEB and ILIKEB predicates do byte-at-a-time ASCII comparisons.

Locale Dependencies

In the default locale, LIKE and ILIKE handle UTF-8 character-at-a-time, locale-insensitive comparisons. ILIKE handles language-independent case-folding.

In non-default locales, LIKE and ILIKE perform locale-sensitive string comparisons, including some automatic normalization, using the same algorithm as the "=" operator on VARCHAR types.

ESCAPE expressions evaluate to exactly one octet—or one UTF-8 character for non-default locales.

Examples

The following example illustrates pattern matching in locales.

```
\locale default=> CREATE TABLE src(c1 VARCHAR(100));  
=> INSERT INTO src VALUES (U&'\00DF'); --The sharp s (ß)  
=> INSERT INTO src VALUES ('ss');  
=> COMMIT;
```

Querying the src table in the default locale returns both ss and sharp s.

```
=> SELECT * FROM src;  
c1  
----  
ß  
ss  
(2 rows)
```

The following query combines pattern-matching predicates to return the results from column c1:

```
=> SELECT c1, c1 = 'ss' AS equality, c1 LIKE 'ss'  
AS LIKE, c1 ILIKE 'ss' AS ILIKE FROM src;  
c1 | equality | LIKE | ILIKE
```

```

-----+-----+-----+-----
ß | f      | f      | f
ss | t      | t      | t
(2 rows)

```

The next query specifies unicode format for c1:

```

=> SELECT c1, c1 = U&'\00DF' AS equality,
      c1 LIKE U&'\00DF' AS LIKE,
      c1 ILIKE U&'\00DF' AS ILIKE from src;
c1 | equality | LIKE | ILIKE
-----+-----+-----+-----
ß | t      | t      | t
ss | f      | f      | f
(2 rows)

```

Now change the locale to German with a strength of 1 (ignore case and accents):

```

\locale LDE_S1
=> SELECT c1, c1 = 'ss' AS equality,
      c1 LIKE 'ss' as LIKE, c1 ILIKE 'ss' AS ILIKE from src;
c1 | equality | LIKE | ILIKE
-----+-----+-----+-----
ß | t      | t      | t
ss | t      | t      | t
(2 rows)

```

This example illustrates binary data types with pattern-matching predicates:

```

=> CREATE TABLE t (c BINARY(1));
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
=> SELECT TO_HEX(c) from t;
TO_HEX
-----
00
ff
(2 rows)
select * from t;
c
-----
\000
\377
(2 rows)
=> SELECT c, c = '\000', c LIKE '\000', c ILIKE '\000' from t;
c | ?column? | ?column? | ?column?
-----+-----+-----+-----
\000 | t      | t      | t
\377 | f      | f      | f
(2 rows)
=> SELECT c, c = '\377', c LIKE '\377', c ILIKE '\377' from t;
c | ?column? | ?column? | ?column?
-----+-----+-----+-----
\000 | f      | f      | f
\377 | t      | t      | t
(2 rows)

```

NULL-predicate

Tests for null values.

Syntax

```
value_expression IS [ NOT ] NULL
```

Parameters

<i>value_expression</i>	A column name, literal, or function.
-------------------------	--------------------------------------

Examples

Column name:

```
=> SELECT date_key FROM date_dimension WHERE date_key IS NOT NULL;
date_key
-----
      1
     366
    1462
    1097
       2
       3
       6
       7
       8
...

```

Function:

```
=> SELECT MAX(household_id) IS NULL FROM customer_dimension;
?column?
-----
f
(1 row)

```

Literal:

```
=> SELECT 'a' IS NOT NULL;
?column?
-----
t

```

(1 row)

Hints

Hints are directives that you embed within a query or directed query. They conform to the following syntax:

```
/*+ hint-name[, hint-name]... */
```

Hints are bracketed by comment characters `/*+` and `*/`, which can enclose multiple comma-delimited hints. For example:

```
/*+ DIRECT, LABEL(myLabel) */
```

Restrictions

When embedding hints, be aware of the following restrictions:

- Do not embed spaces in the comment characters `/*` and `*/`.
- In general, spaces are allowed before and after the plus (+) character and *hint-name*; however, some third-party tools do not support spaces embedded inside `/*+`.

Supported Hints

Vertica supports the following hints:

General hints	<code>ALLNODES</code>	Qualifies an <code>EXPLAIN</code> statement to request a query plan that assumes all nodes are active.
	<code>ENABLE_WITH_CLAUSE_MATERIALIZATION</code>	Enables and disables <code>WITH clause</code> materialization for a specific query.
	<code>EARLY_MATERIALIZATION</code>	Specifies early materialization of a table for the current query.
	<code>DIRECT</code>	Specifies to write data directly to disk (ROS); valid only for <code>CREATE [TEMPORARY] TABLE AS</code> ,

		INSERT , MERGE , and UPDATE operations.
	LABEL	Labels a query so you can identify it for profiling and debugging.
Join hints	SYNTACTIC_JOIN	Enforces join order and enables other join hints.
	DISTRIB	Sets the input operations for a distributed join to broadcast, resegment, local, or filter.
	GBYTYPE	Specifies which algorithm— GROUPBY HASH or GROUPBY PIPELINED —the Vertica query optimizer should use to implement a GROUP BY clause.
	JTYPE	Enforces the join type: merge or hash join.
	UTYPE	Specifies how to combine UNION ALL input.
Table hints	PROJS	Specifies one or more projections to use for a queried table.
	SKIP_PROJS	Specifies which projections to avoid using for a queried table.
Directed query hints	IGNORECONST	Maps an input query constant to one or more annotated query constants.
	VERBATIM	Enforces execution of an annotated query exactly as written.

ALLNODES

Qualifies an [EXPLAIN](#) statement to request a query plan that assumes all nodes are active. If you omit this hint, the [EXPLAIN](#) statement produces a query plan that takes into account any nodes that are currently down.

Syntax

```
EXPLAIN /*+ ALLNODES */ ...
```

Examples

In the following example, the ALLNODES hint requests a query plan that assumes all nodes are active.

```
QUERY PLAN DESCRIPTION:
-----

Opt Vertica Options
-----
PLAN_ALL_NODES_ACTIVE

EXPLAIN /*+ALLNODES*/ select * from Emp_Dimension;

Access Path:
+-STORAGE ACCESS for Emp_Dimension [Cost: 125, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.Emp_Dimension_b0
| Materialize: Emp_Dimension.Employee_key, Emp_Dimension.Employee_gender, Emp_Dimension.Courtesy_
title, Emp_Dimension.Employee_first_name, Emp_Dimension.Employee_middle_initial, Emp_
Dimension.Employee_last_name, Emp_Dimension.Employee_age, Emp_Dimension.Employee_birthdate, Emp_
Dimension.Employee_street, Emp_Dimension.Employee_city, Emp_Dimension.Employee_state, Emp_
Dimension.Employee_region, Emp_Dimension.Employee_position
| Execute on: All Nodes
```

AUTO

Specifies to initially load data into WOS. This hint overrides the table's [load method](#).

The following statements support this hint:

- [INSERT](#)
- [MERGE](#)
- [UPDATE](#)
- [CREATE \[TEMPORARY\] TABLE AS](#)

For more information about data load options, see [Choosing a Load Method](#).

Syntax

```
statement-name /*+ AUTO */ ...
```

DIRECT

Specifies to bypass memory (WOS) and write table data directly to disk (ROS). This hint overrides the table's [load method](#), by default set to AUTO.

The following statements support this hint:

- **DELETE**: If used with `/*+DIRECT*/`, stores delete vectors in ROS.
- **INSERT**
- **MERGE**
- **UPDATE**
- **CREATE [TEMPORARY] TABLE AS**

For more information about data load options, see [Choosing a Load Method](#).

Syntax

```
statement-name /*+ DIRECT */ ...
```

DISTRIB

Specifies to the optimizer how to distribute join key data to implement a join.

Syntax

```
...JOIN /*+ DISTRIB(outer-join, inner-join) */
```

Arguments

<i>outer-join</i> <i>inner-join</i>	<p>Specifies how to distribute data on the outer and inner joins:</p> <ul style="list-style-type: none">• L (local): Inner and outer join keys are identically segmented on each node, join locally.• R (resegment): Inner and outer join keys are not identically segmented.
--	--

	<p>Resegment join-key data before implementing the join.</p> <ul style="list-style-type: none">• B (broadcast): Inner and outer join keys are not identically segmented. Broadcast data of this join key to other nodes before implementing the join.• F (filter): Join table is unsegmented. Filter data as needed by the other join key before implementing the join.• A (any): Let the optimizer choose the distribution method that it considers to be most cost-effective.
--	---

Description

The DISTRIB hint specifies to the optimizer how to distribute join key data in order to implement a join. If a specified distribution is not feasible, the optimizer ignores the hint and throws a warning.

The following requirements apply:

- Queries that include the DISTRIB hint must also include the SYNTACTIC_JOIN hint. Otherwise, the optimizer ignores the DISTRIB hint and throws a warning.
- Join syntax must conform with ANSI SQL-92 join conventions.

Examples

In the following query, the join is qualified with a DISTRIB hint of `/*+ DISTRIB(L,R)*/`. This hint tells the optimizer to resegment data of join key `stores.store_key` before joining it to the `sales.store_key` data:

```
SELECT /*+ SYNTACTIC_JOIN */ sales.store_key, stores.store_name, sales.product_description,  
sales.sales_quantity, sales.sale_date  
FROM (store.storeSales AS sales JOIN /*+DISTRIB(L,R),JTYPE(H)*/ store.store_dimension AS stores ON  
(sales.store_key = stores.store_key))  
WHERE (sales.sale_date = '2014-12-01'::date) ORDER BY sales.store_key, sales.sale_date;
```

EARLY_MATERIALIZATION

Specifies early materialization of a table for the current query. A query can include this hint for any number of tables. Typically, the query optimizer delays materialization until late in the query execution process. This hint overrides any choices that the optimizer otherwise would make.

This hint can be useful in cases where late materialization of join inputs precludes other optimizations—for example, pushing aggregation down the joins, or using live aggregate projections. In these cases, qualifying a join input with `EARLY_MATERIALIZATION` can enable the optimizations.

Syntax

```
table-name [ [AS] alias] /*+ EARLY_MATERIALIZATION */
```

ENABLE_WITH_CLAUSE_MATERIALIZATION

Enables and disables [WITH clause](#) materialization for a specific query. Materialization is automatically cleared when the query returns. For more information about materializing WITH clauses, see [Materialization of WITH Clause](#) in Analyzing Data.

Syntax

```
WITH /*+ENABLE_WITH_CLAUSE_MATERIALIZATION*/ with-query...
```

GBYTYPE

Specifies which algorithm—`GROUPBY HASH` or `GROUPBY PIPELINED`—the Vertica query optimizer should use to implement a [GROUP BY](#) clause. If both algorithms are valid for this query, the query optimizer chooses the specified algorithm over the algorithm that the query optimizer might otherwise choose in its query plan.

Syntax

```
...GROUP BY /*+ GBYTYPE( HASH | PIPE ) */
```

Arguments

HASH PIPE	<p>Specifies the GROUP BY algorithm to use:</p> <ul style="list-style-type: none">• HASH: GROUPBY HASH algorithm• PIPE: GROUPBY PIPELINED algorithm <p>Note: Vertica uses the GROUPBY PIPELINED algorithm only if the query and one of its projections comply with GROUP BY PIPELINED requirements. Otherwise, Vertica issues a warning and uses GROUPBY HASH.</p> <p>For more information about both algorithms, see GROUP BY Implementation Options.</p>
-------------	--

Examples

See [Controlling GROUPBY Algorithm Choice](#) in Analyzing Data.

IGNORECONST

In a directed query, maps an input query constant to one or more annotated query constants.

Syntax

```
/*+ IGNORECONST(arg) */
```

Description

IGNORECONST lets you create directed queries that support input queries with various conditions. IGNORECONST requires an integer argument. This argument matches constants in

input and annotated queries that you want the optimizer to ignore.

Examples

In the following example a directed query is created where input and annotated queries set IGNORECONST hints on Employee_city and Employee_position:

```
=> SAVE QUERY SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension
WHERE Employee_city='somewhere' /*+IGNORECONST(1)*/
AND Employee_position='somejob' /*+IGNORECONST(2)*/;
SAVE QUERY

=> CREATE DIRECTED QUERY CUSTOM 'findEmployees' SELECT Employee_first_name, Employee_last_name FROM
EMP_Dimension /*+projs('public.Emp_Dimension_Unseg')*/
WHERE Employee_city='somewhere' /*+IGNORECONST(1)*/
AND Employee_position='somejob' /*+IGNORECONST(2)*/;
CREATE DIRECTED QUERY
```

IGNORECONST pairs two sets of constants:

- IGNORECONST(1) pairs input and annotated query settings for Employee_city.
- IGNORECONST(2) pairs input and annotated query settings for Employee_position.

See Also

[Ignoring Constants in Directed Queries](#)

JTYPE

Specifies the join algorithm as hash or merge.

Syntax

```
...JOIN /*+ JTYPE(join-type) */
```

Arguments

<i>join-type</i>	One of the following arguments: <ul style="list-style-type: none">• H: Hash join
------------------	--

- M: Merge join, valid only if both join inputs are already sorted on the join columns, otherwise Vertica ignores it and throws a warning.

Note: The optimizer relies upon the query or DDL to verify whether input data is sorted, rather than the actual runtime order of the data.

- FM: Forced merge join. Before performing the merge, the optimizer re-sorts the join inputs. Two restrictions apply:
 - This option is valid only for simple join conditions. For example:

```
SELECT /*+ SYNTACTIC_JOIN*/ * FROM x JOIN /*+JTYPE(FM)*/ y ON x.c1 = y.c1;
```

- Join columns must be of the same type and precision or scale. One exception applies: string columns can have different lengths

Description

Use the JTYPE hint to specify the algorithm the optimizer uses to join table data. If specified algorithm is not feasible, the optimizer ignores the hint and throws a warning.

Requirements

- Queries that include the JTYPE hint must also include the SYNTACTIC_JOIN hint. Otherwise, the optimizer ignores the JTYPE hint and throws a warning.
- Join syntax must conform with ANSI SQL-92 join conventions.

LABEL

Assigns a label to a statement in order to identify it for profiling and debugging.

LABEL hints are valid in the following statements:

- [DELETE](#)
- [INSERT](#)
- [MERGE](#)
- [SELECT](#)

- [UPDATE](#)
- [UNION](#): Valid in the UNION's first SELECT statement. Vertica ignores labels in subsequent SELECT statements.

Syntax

```
statement-name /*+ LABEL (label-string) */
```

Arguments

<i>label-string</i>	A string that is up to 128 octets long. If enclosed with single quotes, <i>label-string</i> can contain embedded spaces.
---------------------	--

Examples

```
=> SELECT /*+ LABEL(MySelectQuery)*/ COUNT(*) FROM t;  
=> INSERT /*+ LABEL('Yet Another Query')*/ INTO t VALUES(1);
```

See Also

[Labeling Queries](#) in the Administrator's Guide

PROJS

Specifies one or more projections to use for a queried table.

Syntax

```
...FROM table-name /*+ PROJS( [schema.]projection[,...] ) */
```

Arguments

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>projection</i>	<p>The projection to use. You can specify a list of comma-delimited projections.</p>

Description

The PROJS hint can specify multiple projections; the optimizer determines which ones are valid and uses the one that is most cost-effective for the queried table. If no hinted projection is valid, the query returns a warning and ignores projection hints.

Examples

The following query includes a PROJS hint that specifies two projections:

```
=> EXPLAIN SELECT * FROM Emp_Dimension /*+PROJS('public.Emp_Dimension_Unseg', 'public.Emp_Dimension')*/;
```

The first projection `public.Emp_Dimension_Unseg` does not include all columns in the queried table `Emp_Dimension`, so the optimizer cannot use it. The second projection includes all table columns so the optimizer uses it, as verified by the following query plan:

```
QUERY PLAN DESCRIPTION:
-----
explain SELECT * FROM Emp_Dimension /*+PROJS('public.Emp_Dimension_Unseg', 'public.Emp_Dimension')*/;
Access Path:
+-STORAGE ACCESS for Emp_Dimension [Cost: 125, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.Emp_Dimension_b0
```

SKIP_PROJS

Specifies which projections to avoid using for a queried table.

Syntax

```
...FROM table-name /*+ SKIP_PROJS( [[db-name.]schema.]projection[,...] ) */
```

Arguments

<code>[<i>db-name</i>.]<i>schema</i></code>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<code><i>projection</i></code>	A projection to skip. You can specify a list of comma-delimited projections.

Description

The SKIP_PROJS specifies one or more projections that the optimizer should avoid using. If the SKIP_PROJS hint excludes all available projections that are valid for the query, the optimizer issues a warning and ignores the projection hints.

Examples

In this example, the EXPLAIN output shows that the optimizer uses the projection `public.Emp_Dimension_b0` for a given query:

```
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT Employee_last_name, Employee_first_name, Employee_city, Employee_position FROM Emp_
Dimension;

Access Path:
+-STORAGE ACCESS for Emp_Dimension [Cost: 59, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.Emp_Dimension_b0
```

You can use the SKIP_PROJS hint to avoid using this projection. If another projection is available that is valid for this query, the optimizer uses it instead:

```
QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT Employee_last_name, Employee_first_name, Employee_city, Employee_position FROM Emp_
Dimension /*+SKIP_PROJS('public.Emp_Dimension')*/;

Access Path:
+-STORAGE ACCESS for Emp_Dimension [Cost: 152, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Projection: public.Emp_Dimension_Unseg
```

SYNTACTIC_JOIN

Enforces join order and enables other join hints.

Syntax

```
/*+ SYN[TACTIC]_JOIN */
```

Description

In order to achieve optimal performance, the optimizer often overrides a query's specified join order. By including the `SYNTACTIC_JOIN` hint, you can ensure that the optimizer enforces the query's join order exactly as specified. One requirement applies: the join syntax must conform with ANSI SQL-92 conventions.

The `SYNTACTIC_JOIN` hint must immediately follow `SELECT`. If the annotated query includes another hint that must also follow `SELECT`, such as `VERBATIM`, combine the two hints together. For example:

```
SELECT /*+ syntactic_join,verbatim */ ...
```

Examples

In the following examples, the optimizer produces different plans for two queries that differ only by including or excluding the `SYNTACTIC_JOIN` hint.

Excludes `SYNTACTIC_JOIN`

```
EXPLAIN SELECT sales.store_key, stores.store_name, products.product_description, sales.sales_
quantity, sales.sale_date
FROM (store.store_sales sales JOIN products ON sales.product_key=products.product_key)
JOIN store.store_dimension stores ON sales.store_key=stores.store_key
WHERE sales.sale_date='2014-12-01' order by sales.store_key, sales.sale_date;
```

```
Access Path:
+-SORT [Cost: 14K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
```

```

| +---> JOIN HASH [Cost: 11K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Outer (RESEGMENT)(LOCAL ROUND
ROBIN) Inner (RESEGMENT)
| |      Join Cond: (sales.product_key = products.product_key)
| |      Materialize at Input: sales.store_key, sales.product_key, sales.sale_date, sales.sales_
quantity
| |      Execute on: All Nodes
| | +-- Outer -> JOIN HASH [Cost: 1K, Rows: 100K (NO STATISTICS)] (PATH ID: 3)
| | |      Join Cond: (sales.store_key = stores.store_key)
| | |      Execute on: All Nodes
| | | +-- Outer -> STORAGE ACCESS for sales [Cost: 1K, Rows: 100K (NO STATISTICS)] (PATH ID: 4)
| | | |      Projection: store.store_sales_b0
| | | |      Materialize: sales.store_key
| | | |      Filter: (sales.sale_date = '2014-12-01'::date)
| | | |      Execute on: All Nodes
| | | |      Runtime Filter: (SIP1(HashJoin): sales.store_key)
| | | +-- Inner -> STORAGE ACCESS for stores [Cost: 34, Rows: 250] (PATH ID: 5)
| | | |      Projection: store.store_dimension_DBD_10_rep_VMartDesign_node0001
| | | |      Materialize: stores.store_key, stores.store_name
| | | |      Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for products [Cost: 3K, Rows: 60K (NO STATISTICS)] (PATH ID: 6)
| | |      Projection: public.products_b0
| | |      Materialize: products.product_key, products.product_description
| | |      Execute on: All Nodes

```

Includes SYNTACTIC_JOIN

```

EXPLAIN SELECT /*+SYNTACTIC_JOIN*/ sales.store_key, stores.store_name, products.product_description,
sales.sales_quantity, sales.sale_date
FROM (store.store_sales sales JOIN products ON sales.product_key=products.product_key)
JOIN store.store_dimension stores ON sales.store_key=stores.store_key
WHERE sales.sale_date='2014-12-01' order by sales.store_key, sales.sale_date;

```

Access Path:

```

+-SORT [Cost: 11K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +---> JOIN HASH [Cost: 8K, Rows: 100K (NO STATISTICS)] (PATH ID: 2)
| |      Join Cond: (sales.store_key = stores.store_key)
| |      Execute on: All Nodes
| | +-- Outer -> JOIN HASH [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 3) Outer (BROADCAST)
(LOCAL ROUND ROBIN)
| | |      Join Cond: (sales.product_key = products.product_key)
| | |      Execute on: All Nodes
| | |      Runtime Filter: (SIP1(HashJoin): sales.store_key)
| | | +-- Outer -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 4)
| | | |      Projection: store.store_sales_b0
| | | |      Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | |      Filter: (sales.sale_date = '2014-12-01'::date)
| | | |      Execute on: All Nodes
| | | +-- Inner -> STORAGE ACCESS for products [Cost: 3K, Rows: 60K (NO STATISTICS)] (PATH ID: 5)
| | | |      Projection: public.products_b0
| | | |      Materialize: products.product_key, products.product_description
| | | |      Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for stores [Cost: 34, Rows: 250] (PATH ID: 6)
| | |      Projection: store.store_dimension_DBD_10_rep_VMartDesign_node0001
| | |      Materialize: stores.store_key, stores.store_name
| | |      Execute on: All Nodes

```

TRICKLE

Specifies to load data only into WOS. This hint overrides the table's [load method](#), by default set to AUTO.

The following statements support this hint:

- [DELETE](#): If used with `/*+TRICKLE*/`, stores delete vectors in WOS.
- [INSERT](#)
- [MERGE](#)
- [UPDATE](#)
- [CREATE \[TEMPORARY\] TABLE AS](#)

For more information about data load options, see [Choosing a Load Method](#).

Syntax

```
statement-name /*+ TRICKLE */ ...
```

UTYPE

Specifies how to combine [UNION ALL](#) input.

Syntax

```
...UNION ALL /*+ UTYPE(union-type) */
```

Arguments

<i>union-type</i>	One of the following arguments: <ul style="list-style-type: none">• U: Concatenates UNION ALL input (default).• M: Merges UNION ALL input in the same sort order as the source query results. This option requires all input from the source queries to use the
-------------------	--

same sort order; otherwise, Vertica throws a warning and concatenates the UNION ALL input.

Note: The optimizer relies upon the query or DDL to verify whether input data is sorted, rather than the actual runtime order of the data.

Requirements

Queries that include the UTYPE hint must also include the SYNTACTIC_JOIN hint. Otherwise, the optimizer ignores the UTYPE hint and throws a warning.

VERBATIM

Enforces execution of an annotated query exactly as written.

VERBATIM directs the optimizer to create a query plan that incorporates all hints in a annotated query. Furthermore, it directs the optimizer not to apply its own plan development processing on query plan components that pertain to those hints.

Usage of this hint varies between [optimizer-generated](#) and [custom](#) directed queries, as described below.

Syntax

```
SELECT /*+ VERBATIM */ ...
```

Requirements

The VERBATIM hint must immediately follow SELECT. If the annotated query includes another hint that must also follow SELECT, such as SYNTACTIC_JOIN, combine the two hints together. For example:

```
SELECT /*+ syntactic_join,verbatim */ ...
```

Optimizer-Generated Directed Queries

The optimizer always includes the VERBATIM hint in the annotated queries that it [generates for directed queries](#). For example, given the following

CREATE DIRECTED QUERY OPTIMIZER statement:

```
=> CREATE DIRECTED QUERY OPTIMIZER getStoreSales SELECT sales.store_key, stores.store_name,
sales.product_description, sales.sales_quantity, sales.sale_date FROM store.storesales sales JOIN
store.store_dimension stores ON sales.store_key=stores.store_key WHERE sales.sale_date='2014-12-01'
/*+IGNORECONST(1)*/ AND stores.store_name='Store1' /*+IGNORECONST(2)*/ ORDER BY sales.store_key,
sales.sale_date;
CREATE DIRECTED QUERY
```

The optimizer generates an annotated query that includes the VERBATIM hint:

```
=> SELECT query_name, annotated_query FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name =
'getStoreSales';
-[ RECORD 1 ]-----
query_name      | getStoreSales
annotated_query | SELECT /*+ syntactic_join,verbatim */ sales.store_key AS store_key, stores.store_
name AS store_name, sales.product_description AS product_description, sales.sales_quantity AS sales_
quantity, sales.sale_date AS sale_date
FROM (store.storeSales AS sales/*+projs('store.storeSales')*/ JOIN /*+Distrib(L,L),JType(H)*/
store.store_dimension AS stores/*+projs('store.store_dimension_DBD_10_rep_VMartDesign')*/ ON
(sales.store_key = stores.store_key))
WHERE (sales.sale_date = '2014-12-01'::date /*+IgnoreConst(1)*/) AND (stores.store_name =
'Store1'::varchar(6) /*+IgnoreConst(2)*/)
ORDER BY 1 ASC, 5 ASC
```

When the optimizer uses this directed query, it produces a query plan that is equivalent to the query plan that it used when it created the directed query:

```
=> ACTIVATE DIRECTED QUERY getStoreSales;
ACTIVATE DIRECTED QUERY

=> EXPLAIN SELECT sales.store_key, stores.store_name, sales.product_description, sales.sales_
quantity, sales.sale_date FROM store.storesales sales JOIN store.store_dimension stores ON
sales.store_key=stores.store_key WHERE sales.sale_date='2014-12-04' AND stores.store_name='Store14'
ORDER BY sales.store_key, sales.sale_date;

QUERY PLAN DESCRIPTION:
-----

EXPLAIN SELECT sales.store_key, stores.store_name, sales.product_description, sales.sales_quantity,
sales.sale_date FROM store.storesales sales JOIN store.store_dimension stores ON sales.store_
key=stores.store_key WHERE sales.sale_date='2014-12-04' AND stores.store_name='Store14' ORDER BY
sales.store_key, sales.sale_date;

The following active directed query(query name: getStoreSales) is being executed:
SELECT /*+syntactic_join,verbatim*/ sales.store_key, stores.store_name, sales.product_description,
sales.sales_quantity, sales.sale_date
FROM (store.storeSales sales/*+projs('store.storeSales')*/ JOIN /*+Distrib('L', 'L'), JType
('H')*/store.store_dimension stores
```

```
/*+projs('store.store_dimension_DBD_10_rep_VMartDesign')*/ ON ((sales.store_key = stores.store_key)))  
WHERE ((sales.sale_date = '2014-12-04'::date)  
AND (stores.store_name = 'Store14'::varchar(7))) ORDER BY sales.store_key, sales.sale_date
```

Access Path:

```
+--JOIN HASH [Cost: 463, Rows: 622 (NO STATISTICS)] (PATH ID: 2)  
| Join Cond: (sales.store_key = stores.store_key)  
| Materialize at Output: sales.sale_date, sales.sales_quantity, sales.product_description  
| Execute on: All Nodes  
| +-- Outer -> STORAGE ACCESS for sales [Cost: 150, Rows: 155K (NO STATISTICS)] (PATH ID: 3)  
| | Projection: store.storeSales_b0  
| | Materialize: sales.store_key  
| | Filter: (sales.sale_date = '2014-12-04'::date)  
| | Execute on: All Nodes  
| | Runtime Filter: (SIP1(HashJoin): sales.store_key)  
| +-- Inner -> STORAGE ACCESS for stores [Cost: 35, Rows: 2] (PATH ID: 4)  
| | Projection: store.store_dimension_DBD_10_rep_VMartDesign_node0001  
| | Materialize: stores.store_name, stores.store_key  
| | Filter: (stores.store_name = 'Store14')  
| | Execute on: All Nodes
```

Custom Directed Queries

The VERBATIM hint is included in a [custom directed query](#) only if you explicitly include it in the annotated query that you write for that directed query. When the optimizer uses that directed query, it respects the VERBATIM hint and creates a query plan accordingly.

If you omit the VERBATIM hint when you create a custom directed query, the hint is not stored with the annotated query. When the optimizer uses that directed query, it applies its own plan development processing on the annotated query before it generates a query plan. This query plan might not be equivalent to the query plan that the optimizer would have generated for the Vertica version in which the directed query was created.

SQL Data Types

The following table summarizes the data types that Vertica supports. It also shows the default placement of null values in projections. The Size column is listed as uncompressed bytes.

Type	Size (bytes)	Description	NULL Sorting
Binary types (For more information, see Binary Data Types .)			
BINARY	1 to 65000	Fixed-length binary string	NULLS LAST
VARBINARY	1 to 65000	Variable-length binary string	NULLS LAST
LONG VARBINARY	1 to 32,000,000	Long variable-length binary string	NULLS LAST
BYTEA	1 to 65000	Variable-length binary string (synonym for VARBINARY)	NULLS LAST
RAW	1 to 65000	Variable-length binary string (synonym for VARBINARY)	NULLS LAST
Boolean types (See Boolean Data Type .)			
BOOLEAN	1	True or False or NULL	NULLS LAST
Character types (See Character Data Types and Long Data Types .)			
CHAR	1 to 65000	Fixed-length character string	NULLS LAST
VARCHAR	1 to 65000	Variable-length character string	NULLS LAST
LONG VARCHAR	1 to 32,000,000	Long variable-length character string	NULLS LAST
Date/time types (See Date/Time Data Types .)			
DATE	8	Represents a month, day, and year	NULLS FIRST
DATETIME	8	Represents a date and time with or without timezone (synonym for TIMESTAMP)	NULLS FIRST
SMALLDATETIME	8	Represents a date and time with or	NULLS FIRST

Type	Size (bytes)	Description	NULL Sorting
		without timezone (synonym for <code>TIMESTAMP</code>)	
<code>TIME</code>	8	Represents a time of day without timezone	NULLS FIRST
<code>TIME WITH TIMEZONE</code>	8	Represents a time of day with timezone	NULLS FIRST
<code>TIMESTAMP</code>	8	Represents a date and time without timezone	NULLS FIRST
<code>TIMESTAMP WITH TIMEZONE</code>	8	Represents a date and time with timezone	NULLS FIRST
<code>INTERVAL</code>	8	Measures the difference between two points in time	NULLS FIRST
<code>INTERVAL DAY TO SECOND</code>	8	Represents an interval measured in days and seconds	NULLS FIRST
<code>INTERVAL YEAR TO MONTH</code>	8	Represents an interval measured in years and months	NULLS FIRST
Approximate numeric types (See DOUBLE PRECISION (FLOAT) .)			
<code>DOUBLE PRECISION</code>	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
<code>FLOAT</code>	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
<code>FLOAT(n)</code>	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
<code>FLOAT8</code>	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
<code>REAL</code>	8	Signed 64-bit IEEE floating point number, requiring 8 bytes of storage	NULLS LAST
Exact numeric types (See Numeric Data Types .)			

Type	Size (bytes)	Description	NULL Sorting
INTEGER	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
INT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
BIGINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
INT8	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
SMALLINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
TINYINT	8	Signed 64-bit integer, requiring 8 bytes of storage	NULLS FIRST
DECIMAL	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
NUMERIC	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
NUMBER	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST
MONEY	8+	8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits	NULLS FIRST

Binary Data Types

Store raw-byte data, such as IP addresses, up to 65000 bytes.

Syntax

```
BINARY ( Length ) { VARBINARY | BINARY VARYING | BYTEA | RAW } ( max-Length )
```

Parameters

<i>Length</i> <i>max-Length</i>	Specifies the length of the string (column width, declared in bytes (octets), in CREATE TABLE statements).
-----------------------------------	--

Notes

- BYTEA and RAW are synonyms for VARBINARY.
- The data types BINARY and BINARY VARYING (VARBINARY) are collectively referred to as *binary string types* and the values of binary string types are referred to as *binary strings*.
- A binary string is a sequence of octets, or bytes. Binary strings store raw-byte data, while character strings store text.
- A binary value value of NULL appears last (largest) in ascending order.
- The binary data types, BINARY and VARBINARY, are similar to the [Character Data Types](#), CHAR and VARCHAR, respectively, except that binary data types contain byte strings, rather than character strings.
- BINARY—A fixed-width string of *length* bytes, where the number of bytes is declared as an optional specifier to the type. If length is omitted, the default is 1. Where necessary, values are right-extended to the full width of the column with the zero byte. For example:

```
=> SELECT TO_HEX('ab'::BINARY(4));  
to_hex  
-----  
61620000
```

- VARBINARY—A variable-width string up to a length of *max-length* bytes, where the maximum number of bytes is declared as an optional specifier to the type. The default is the default attribute size, which is 80, and the maximum length is 65000 bytes. VARBINARY values are not extended to the full width of the column. For example:

```
=> SELECT TO_HEX('ab'::VARBINARY(4));  
to_hex  
-----  
6162
```

- You can use several formats when working with binary values, but the hexadecimal format is generally the most straightforward and is emphasized in Vertica documentation.
- Binary operands `&`, `~`, `|` and `#` have special behavior for binary data types, as described in [Binary Operators](#).
- On input, strings are translated from:
 - Hexadecimal representation to a binary value using the [HEX_TO_BINARY](#) function
 - Bitstring representation to a binary value using the [BITSTRING_TO_BINARY](#) function.

Both functions take a `VARCHAR` argument and return a `VARBINARY` value. See the Examples section below.

- Binary values can also be represented in octal format by prefixing the value with a backslash `'\'`.

Note: If you use `vsql`, you must use the escape character (`\`) when you insert another backslash on input; for example, input `'\141'` as `'\\141'`.

You can also input values represented by printable characters. For example, the hexadecimal value `'0x61'` can also be represented by the symbol `'1'`.

See [Bulk-Loading Data](#) in the Administrator's Guide.

- Like the input format the output format is a hybrid of octal codes and printable ASCII characters. A byte in the range of printable ASCII characters (the range `[0x20, 0x7e]`) is represented by the corresponding ASCII character, with the exception of the backslash (`'\'`), which is escaped as `'\\'`. All other byte values are represented by their corresponding octal values. For example, the bytes `{97,92,98,99}`, which in ASCII are `{a, \, b, c}`, are translated to text as `'a\\bc'`.
- The following aggregate functions are supported for binary data types:
 - [BIT_AND](#)
 - [BIT_OR](#)

- [BIT_XOR](#)
- [MAX](#)
- [MIN](#)

`BIT_AND`, `BIT_OR`, and `BIT_XOR` are bitwise operations that are applied to each non-null value in a group, while `MAX` and `MIN` are bitwise comparisons of binary values.

- Like their [binary operator](#) counterparts, if the values in a group vary in length, the aggregate functions treat the values as though they are all equal in length by extending shorter values with zero bytes to the full width of the column. For example, given a group containing the values `'ff'`, `null`, and `'f'`, a binary aggregate ignores the null value and treats the value `'f'` as `'f0'`. Also, like their binary operator counterparts, these aggregate functions operate on `VARBINARY` types explicitly and operate on `BINARY` types implicitly through casts. See [Data Type Coercion Operators \(CAST\)](#).

Examples

The following example shows `VARBINARY` [HEX_TO_BINARY](#)(`VARCHAR`) and `VARCHAR` [TO_HEX](#)(`VARBINARY`) usage.

Table `t` and its projection are created with binary columns:

```
=> CREATE TABLE t (c BINARY(1));
=> CREATE PROJECTION t_p (c) AS SELECT c FROM t;
```

Insert minimum byte and maximum byte values:

```
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
```

Binary values can then be formatted in hex on output using the `TO_HEX` function:

```
=> SELECT TO_HEX(c) FROM t;
to_hex
-----
00
ff
(2 rows)
```

The `BIT_AND`, `BIT_OR`, and `BIT_XOR` functions are interesting when operating on a group of values. For example, create a sample table and projections with binary columns:

The example that follows uses table `t` with a single column of `VARBINARY` data type:

```
=> CREATE TABLE t ( c VARBINARY(2) );  
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table t to see column c output:

```
=> SELECT TO_HEX(c) FROM t;  
TO_HEX  
-----  
ff00  
ffff  
f00f  
(3 rows)
```

Now issue the bitwise AND operation. Because these are aggregate functions, an implicit GROUP BY operation is performed on results using $(ff00 \& (ffff) \& f00f)$:

```
=> SELECT TO_HEX(BIT_AND(c)) FROM t;  
TO_HEX  
-----  
f000  
(1 row)
```

Issue the bitwise OR operation on $(ff00 | (ffff) | f00f)$:

```
=> SELECT TO_HEX(BIT_OR(c)) FROM t;  
TO_HEX  
-----  
ffff  
(1 row)
```

Issue the bitwise XOR operation on $(ff00 \# (ffff) \# f00f)$:

```
=> SELECT TO_HEX(BIT_XOR(c)) FROM t;  
TO_HEX  
-----  
f0f0  
(1 row)
```

See Also

- [BIT_AND](#)
- [BIT_OR](#)
- [BIT_XOR](#)
- [MAX \[Aggregate\]](#)

- [MIN \[Aggregate\]](#)
- [Binary Operators](#)
- [COPY](#)
- [Data Type Coercion Operators \(CAST\)](#)
- [INET_ATON](#)
- [INET_NTOA](#)
- [V6_ATON](#)
- [V6_NTOA](#)
- [V6_SUBNETA](#)
- [V6_SUBNETN](#)
- [V6_TYPE](#)
- [BITCOUNT](#)
- [BITSTRING_TO_BINARY](#)
- [HEX_TO_BINARY](#)
- [LENGTH](#)
- [REPEAT](#)
- [SUBSTRING](#)
- [TO_HEX](#)
- [TO_BITSTRING](#)

Boolean Data Type

Vertica provides the standard SQL type `BOOLEAN`, which has two states: `true` and `false`. The third state in SQL boolean logic is `unknown`, which is represented by the `NULL` value.

Syntax

BOOLEAN

Parameters

Valid literal data values for input are:

TRUE	't'	'true'	'y'	'yes'	'1'	1
FALSE	'f'	'false'	'n'	'no'	'0'	0

Notes

- Do not confuse the BOOLEAN data type with [Boolean Operators](#) or the [Boolean-Predicate](#).
- The keywords TRUE and FALSE are preferred and are SQL-compliant.
- A Boolean value of NULL appears last (largest) in ascending order.
- All other values must be enclosed in single quotes.
- Boolean values are output using the letters t and f.

See Also

- [NULL Value](#)
- [Data Type Coercion Chart](#)

Character Data Types

Stores strings of letters, numbers, and symbols.

Character data can be stored as fixed-length or variable-length strings. Fixed-length strings are right-extended with spaces on output; variable-length strings are not extended.

Syntax

```
[ CHARACTER | CHAR ] ( octet_length ) [ VARCHAR | CHARACTER VARYING ] ( octet_length )
```

Parameters

<i>octet_length</i>	Specifies the length of the string (column width, declared in bytes (octets), in CREATE TABLE statements).
---------------------	--

Notes

- The data types CHARACTER (CHAR) and CHARACTER VARYING (VARCHAR) are collectively referred to as *character string types*, and the values of character string types are known as *character strings*.
- CHAR is conceptually a fixed-length, blank-padded string. Any trailing blanks (spaces) are removed on input, and only restored on output. The default length is 1, and the maximum length is 65000 octets (bytes).
- VARCHAR is a variable-length character data type. The default length is 80, and the maximum length is 65000 octets. For string values longer than 65000, use [Long Data Types](#). Values can include trailing spaces.
- Normally, you use VARCHAR for all of your string data. You only use CHAR when you need fixed-width string output. For example, you can use CHAR columns for data to be transferred to a legacy system that requires fixed-width strings.
- When you define character columns, specify the maximum size of any string to be stored in a column. For example, to store strings up to 24 octets in length, use either of the following definitions:

```
CHAR(24)      /* fixed-length */VARCHAR(24) /* variable-length */
```

- The maximum length parameter for VARCHAR and CHAR data type refers to the number of octets that can be stored in that field, not the number of characters (Unicode code points). When using multibyte UTF-8 characters, the fields must be sized to accommodate from 1 to

4 octets per character, depending on the data. If the data loaded into a VARCHAR/CHAR column exceeds the specified maximum size for that column, data is truncated on UTF-8 character boundaries to fit within the specified size. See COPY.

Note: Remember to include the extra octets required for multibyte characters in the column-width declaration, keeping in mind the 65000 octet column-width limit.

- String literals in SQL statements must be enclosed in single quotes.
- Due to compression in Vertica, the cost of overestimating the length of these fields is incurred primarily at load time and during sorts.
- NULL appears last (largest) in ascending order. See also [GROUP BY Clause](#) for additional information about NULL ordering.

The Difference Between NULL and NUL

NUL represents a character whose ASCII/Unicode code is 0, sometimes qualified "ASCII NUL".

NULL means no value, and is true of a field (column) or constant, not of a character.

CHAR, LONG VARCHAR, and VARCHAR string data types accept ASCII NULs.

The following example casts the input string containing NUL values to VARCHAR:

```
=> SELECT 'vert\0ica'::CHARACTER VARYING AS VARCHAR;  
VARCHAR  
-----  
vert\0ica  
(1 row)
```

The result contains 9 characters:

```
=> SELECT LENGTH('vert\0ica'::CHARACTER VARYING);  
length  
-----  
9  
(1 row)
```

If you use an [extended string literal](#), the length is 8 characters:

```
=> SELECT E'vert\0ica'::CHARACTER VARYING AS VARCHAR;  
VARCHAR  
-----  
vertica  
(1 row)  
=> SELECT LENGTH(E'vert\0ica'::CHARACTER VARYING);
```

```
LENGTH
-----
      8
(1 row)
```

See Also

- [Data Type Coercion](#)

Date/Time Data Types

Vertica supports the full set of SQL date and time data types. In most cases, a combination of DATE, DATETIME, SMALLDATETIME, TIME, TIMESTAMP WITHOUT TIME ZONE, and TIMESTAMP WITH TIME ZONE, and INTERVAL provides a complete range of date/time functionality required by any application.

In compliance with the SQL standard, Vertica also supports the TIME WITH TIME ZONE data type.

The following table lists the characteristics about the date/time data types. All these data types have a size of 8 bytes.

Name	Description	Low Value	High Value	Resolution
DATE	Dates only (no time of day)	~ 25e+15 BC	~ 25e+15 AD	1 day
TIME [(p)]	Time of day only (no date)	00:00:00.00	23:59:60.999999	1 μ s
TIMETZ [(p)]	Time of day only, with time zone	00:00:00.00+14	23:59:59.999999-14	1 μ s
TIMESTAMP [(p)]	Both date and time, without time zone	290279-12-22 19:59:05.224194 BC	294277-01-09 04:00:54.775806 AD	1 μ s
TIMESTAMPZ [(p)]	Both date and time, with time zone	290279-12-22 19:59:05.224194 BC UTC	294277-01-09 04:00:54.775806 AD UTC	1 μ s
INTERVAL [(p)]DAY TO SECOND	Time intervals	-106751991 days 04:00:54.775807	+106751991 days 04:00:54.775807	1 μ s
INTERVAL [(p)]YEAR TO MONTH	Time intervals	~ -768e15 yrs	~ 768e15 yrs	1 month

Time Zone Abbreviations for Input

Vertica recognizes the files in /opt/vertica/share/timezonesets as date/time input values and defines the default list of strings accepted in the AT TIME ZONE zone parameter. The names are not necessarily used for date/time output—output is driven by the official time zone abbreviations associated with the currently selected time zone parameter setting.

Notes

- In Vertica, `TIME ZONE` is a synonym for `TIMEZONE`.
- Vertica uses Julian dates for all date/time calculations, which can correctly predict and calculate any date more recent than 4713 BC to far into the future, based on the assumption that the average length of the year is 365.2425 days.
- All date/time types are stored in eight bytes.
- A date/time value of `NULL` appears first (smallest) in ascending order.
- All the date/time data types accept the special literal value `NOW` to specify the current date and time. For example:

```
=> SELECT TIMESTAMP 'NOW';
      ?column?
-----
2012-03-13 11:42:22.766989
(1 row)
```

- In Vertica, the [INTERVAL](#) data type is SQL:2008 compliant and allows modifiers, called [interval qualifiers](#), that divide the `INTERVAL` type into two primary subtypes, `DAY TO SECOND` (the default) and `YEAR TO MONTH`. You use the [SET INTERVALSTYLE](#) command to change the `intervalstyle` run-time parameter for the current session.

Intervals are represented internally as some number of microseconds and printed as up to 60 seconds, 60 minutes, 24 hours, 30 days, 12 months, and as many years as necessary. Fields can be positive or negative.

See Also

- [TZ Environment Variable](#)
- [Using Time Zones With Vertica](#)
- [Sources for Time Zone and Daylight Saving Time Data](#)

DATE

Consists of a month, day, and year.

Syntax

DATE

Parameters/Limits

Low Value	High Value	Resolution
~ 25e+15 BC	~ 25e+15 AD	1 DAY

See [SET DATESTYLE](#) for information about ordering.

Example	Description
January 8, 1999	Unambiguous in any datestyle input mode
1999-01-08	ISO 8601; January 8 in any mode (recommended format)
1/8/1999	January 8 in MDY mode; August 1 in DMY mode
1/18/1999	January 18 in MDY mode; rejected in other modes
01/02/03	January 2, 2003 in MDY mode February 1, 2003 in DMY mode February 3, 2001 in YMD mode
1999-Jan-08	January 8 in any mode
Jan-08-1999	January 8 in any mode
08-Jan-1999	January 8 in any mode
99-Jan-08	January 8 in YMD mode, else error
08-Jan-99	January 8, except error in YMD mode
Jan-08-99	January 8, except error in YMD mode

Example	Description
19990108	ISO 8601; January 8, 1999 in any mode
990108	ISO 8601; January 8, 1999 in any mode
1999.008	Year and day of year
J2451187	Julian day
January 8, 99 BC	Year 99 before the Common Era

DATETIME

DATETIME is an alias for [TIMESTAMP/TIMESTAMPTZ](#).

INTERVAL

Measures the difference between two points in time. Intervals can be positive or negative. The INTERVAL data type is divided into two major subtypes:

- [Year-month](#): Span of years and months
- [Day-time](#): Span of days, hours, minutes, seconds, and fractional seconds

Syntax

```
INTERVAL [ ( p ) ] 'interval-literal' [ interval-qualifier ]
```

Parameters

<i>p</i>	Specifies precision of the seconds field, where <i>p</i> is an integer between 0 - 6. For details, see Specifying Interval Precision . Default: 6
<i>interval-literal</i>	A character string that expresses an interval, conforms to this format: <code>[-] { <i>quantity subtype-unit</i> } [...] [AGO]</code>

	For details, see Interval Literal .
<i>interval-qualifier</i>	Optionally specifies how to interpret and format an interval literal for output, and, optionally, sets precision. If omitted, the default is DAY TO SECOND(6). For details, see Interval Qualifier .

Limits

Name	Low Value	High Value	Resolution
INTERVAL [(p)] DAY TO SECOND	-106751991 days 04:00:54.775807	+/-106751991 days 04:00:54.775807	1 microsecond
INTERVAL [(p)] YEAR TO MONTH	~/ -768e15 yrs	~ 768e15 yrs	1 month

Setting Interval Unit Display

You can control how Vertica displays interval units in a SELECT INTERVAL query, with [SET INTERVALSTYLE](#) and [SET DATESTYLE](#). These statements only affect interval output format.

Important: DATESTYLE settings supersede INTERVALSTYLE. If DATESTYLE is set to SQL, interval unit display always conforms to the SQL:2008 standard, which omits interval unit display. If DATESTYLE is set to ISO, you can use [SET INTERVALSTYLE](#) to omit or display interval unit display, as described below.

Omitting Interval Units

To omit interval units from the output, set INTERVALSTYLE to PLAIN. This is the default setting, which conforms with the SQL:2008 standard:

```
=> SET INTERVALSTYLE TO PLAIN;
SET
=> SELECT INTERVAL '3 2';
?column?
-----
3 02:00
```

When `INTERVALSTYLE` is set to `PLAIN`, units are omitted from the output, even if the query specifies input units:

```
=> SELECT INTERVAL '3 days 2 hours';  
?column?  
-----  
3 02:00
```

If `DATESTYLE` is set to `SQL`, Vertica conforms with SQL:2008 standard and always omits interval units from output:

```
=> SET DATESTYLE TO SQL;  
SET  
=> SET INTERVALSTYLE TO UNITS;  
SET  
=> SELECT INTERVAL '3 2';  
?column?  
-----  
3 02:00
```

Displaying Interval Units

To enable display of interval units, `DATESTYLE` must be set to `ISO`. You can then display interval units by setting `INTERVALSTYLE` to `UNITS`:

```
=> SET DATESTYLE TO ISO;  
SET  
=> SET INTERVALSTYLE TO UNITS;  
SET  
=> SELECT INTERVAL '3 2';  
?column?  
-----  
3 days 2 hours
```

Checking INTERVALSTYLE and DATESTYLE Settings

Use `SHOW` statements to check `INTERVALSTYLE` and `DATESTYLE` settings:

```
=> SHOW INTERVALSTYLE;  
name      | setting  
-----+-----  
intervalstyle | units  
=> SHOW DATESTYLE;  
name      | setting  
-----+-----  
datestyle  | ISO, MDY
```

Specifying Interval Input

Interval values are expressed through [interval literals](#). An interval literal is composed of one or more interval fields, where each field represents a span of days and time, or years and months, as follows:

```
[ - ] { quantity subtype-unit } [ ... ] [ AGO ]
```

Using Subtype Units

Subtype units are optional for [day-time](#) intervals; they must be specified for [year-month](#) intervals.

For example, the first statement below implicitly specifies days and time; the second statement explicitly identifies day and time units. Both statements return the same result:

```
=> SET INTERVALSTYLE TO UNITS;
=> SELECT INTERVAL '1 12:59:10:05';
      ?column?
-----
1 day 12:59:10.005
(1 row)

=> SELECT INTERVAL '1 day 12 hours 59 min 10 sec 5 milliseconds';
      ?column?
-----
1 day 12:59:10.005
(1 row)
```

The following two statements add 28 days and 4 weeks to the current date, respectively. The intervals in both cases are equal and the statements return the same result. However, in the first statement, the interval literal omits the subtype (implicitly days); in the second statement, the interval literal must include the subtype unit weeks:

```
=> SELECT CURRENT_DATE;
      ?column?
-----
2016-08-15
(1 row)

=> SELECT CURRENT_DATE + INTERVAL '28';
      ?column?
-----
2016-09-12 00:00:00
(1 row)

dbadmin=> SELECT CURRENT_DATE + INTERVAL '4 weeks';
      ?column?
-----
2016-09-12 00:00:00
```

```
(1 row)
```

An interval literal can include day-time and year-month fields. For example, the following statement adds an interval of 4 years, 4 weeks, 4 days and 14 hours to the current date. The years and weeks fields must include subtype units; the days and hours fields omit them:

```
> SELECT CURRENT_DATE + INTERVAL '4 years 4 weeks 4 14';
      ?column?
-----
2020-09-15 14:00:00
(1 row)
```

Omitting Subtype Units

You can specify quantities of days, hours, minutes, and seconds without specifying units. Vertica recognizes colons in interval literals as part of the timestamp:

```
=> SELECT INTERVAL '1 4 5 6';
      ?column?
-----
1 day 04:05:06
=> SELECT INTERVAL '1 4:5:6';
      ?column?
-----
1 day 04:05:06
=> SELECT INTERVAL '1 day 4 hour 5 min 6 sec';
      ?column?
-----
1 day 04:05:06
```

If Vertica cannot determine the units, it applies the quantity to any missing units based on the interval qualifier. In the next two examples, Vertica uses the default interval qualifier (`DAY TO SECOND(6)`) and assigns the trailing 1 to days, since it has already processed hours, minutes, and seconds in the output:

```
=> SELECT INTERVAL '4:5:6 1';
      ?column?
-----
1 day 04:05:06
=> SELECT INTERVAL '1 4:5:6';
      ?column?
-----
1 day 04:05:06
```

In the next two examples, Vertica recognizes 4:5 as hours:minutes. The remaining values in the interval literal are assigned to the missing units: 1 is assigned to days and 2 is assigned to seconds.

```
SELECT INTERVAL '4:5 1 2';
?column?
-----
1 day 04:05:02
=> SELECT INTERVAL '1 4:5 2';
?column?
-----
1 day 04:05:02
```

Specifying the interval qualifier can change how Vertica interprets 4 : 5:

```
=> SELECT INTERVAL '4:5' MINUTE TO SECOND;
?column?
-----
00:04:05
```

Controlling Interval Format

[Interval qualifiers](#) specify a range of options that Vertica uses to interpret and format an [interval literal](#). The interval qualifier can also specify precision. Each interval qualifier is composed of one or two units:

```
unit[p] [ TO unit[p] ]
```

where:

- *unit* specifies a day-time or year-month [subtype](#).
- *p* specifies precision, an integer between 0 and 6. In general, precision only applies to SECOND units. The default precision for SECOND is 6. For details, see [Specifying Interval Precision](#).

If an interval omits an interval qualifier, Vertica uses the default DAY TO SECOND(6).

Interval Qualifier Categories

Interval qualifiers belong to one of the following categories:

- Year-month: Span of years and months
- Day-time: Span of days, hours, minutes, seconds, and fractional seconds

Year-Month

Vertica supports two year-month subtypes: YEAR and MONTH.

In the following example, YEAR TO MONTH qualifies the interval literal 1 2 to indicate a span of 1 year and two months:

```
=> SELECT interval '1 2' YEAR TO MONTH;
      ?column?
-----
1 year 2 months
(1 row)
```

If you omit the qualifier, Vertica uses the default interval qualifier DAY TO SECOND and returns a different result:

```
=> SELECT interval '1 2';
      ?column?
-----
1 day 2 hours
(1 row)
```

The following example uses the interval qualifier YEAR. In this case, Vertica extracts only the year from the interval literal 1y 10m :

```
=> SELECT INTERVAL '1y 10m' YEAR;
      ?column?
-----
1 year
(1 row)
```

In the next example, the interval qualifier MONTH converts the same interval literal to months:

```
=> SELECT INTERVAL '1y 10m' MONTH;
      ?column?
-----
22 months
(1 row)
```

Day-Time

Vertica supports four day-time subtypes: DAY, HOUR, MINUTE, and SECOND.

In the following example, the interval qualifier DAY TO SECOND(4) qualifies the interval literal 1h 3m 6s 5msecs 57us. The qualifier also sets precision on seconds to 4:

```
=> SELECT INTERVAL '1h 3m 6s 5msecs 57us' DAY TO SECOND(4);
      ?column?
-----
01:03:06.0051
(1 row)
```

If no interval qualifier is specified, Vertica uses the default subtype DAY TO SECOND(6), regardless of how you specify the interval literal. For example, as an extension to SQL:2008, both of the following commands return 910 days:

```
=> SELECT INTERVAL '2-6';
      ?column?
-----
    910 days
=> SELECT INTERVAL '2 years 6 months';
      ?column?
-----
    910 days
```

An interval qualifier can extract other values from the input parameters. For example, the following command extracts the HOUR value from the interval literal `3 days 2 hours`:

```
=> SELECT INTERVAL '3 days 2 hours' HOUR;
      ?column?
-----
    74 hours
```

The primary day/time (`DAY TO SECOND`) and year/month (`YEAR TO MONTH`) subtype ranges can be restricted to more specific range of types by an interval qualifier. For example, `HOUR TO MINUTE` is a limited form of day/time interval, which can be used to express time zone offsets.

```
=> SELECT INTERVAL '1 3' HOUR to MINUTE;
      ?column?
-----
    01:03
```

`hh:mm:ss` and `hh:mm` formats are used only when at least two of the fields specified in the interval qualifier are non-zero and there are no more than 23 hours or 59 minutes:

```
=> SELECT INTERVAL '2 days 12 hours 15 mins' DAY TO MINUTE;
      ?column?
-----
    2 days 12:15
=> SELECT INTERVAL '15 mins 20 sec' MINUTE TO SECOND;
      ?column?
-----
    00:15:20
=> SELECT INTERVAL '1 hour 15 mins 20 sec' MINUTE TO SECOND;
      ?column?
-----
    75 mins 20 secs
```

Specifying Interval Precision

In general, interval precision only applies to seconds. If no precision is explicitly specified, Vertica rounds precision to a maximum of six decimal places. For example:

```
=> SELECT INTERVAL '2 hours 4 minutes 3.709384766 seconds' DAY TO SECOND;
      ?column?
-----
```

```
02:04:03.709385  
(1 row)
```

Vertica lets you specify interval precision in two ways:

- After the `INTERVAL` keyword
- After the `SECOND` unit of an interval qualifier, one of the following:
 - `DAY TO SECOND`
 - `HOUR TO SECOND`
 - `MINUTE TO SECOND`
 - `SECOND`

For example, the following statements use both methods to set precision, and return identical results:

```
=> SELECT INTERVAL(4) '2 hours 4 minutes 3.709384766 seconds' DAY TO SECOND;  
?column?  
-----  
02:04:03.7094  
(1 row)  
  
=> SELECT INTERVAL '2 hours 4 minutes 3.709384766 seconds' DAY TO SECOND(4);  
?column?  
-----  
02:04:03.7094  
(1 row)
```

If the same statement specifies precision more than once, Vertica uses the lesser precision. For example, the following statement specifies precision twice: the `INTERVAL` keyword specifies precision of 1, while the interval qualifier `SECOND` specifies precision of 2. Vertica uses the lesser precision of 1:

```
=> SELECT INTERVAL(1) '1.2467' SECOND(2);  
?column?  
-----  
1.2 secs
```

Setting Precision on Interval Table Columns

If you create a table with an interval column, the following restrictions apply to the column definition:

- You can set precision on the INTERVAL keyword only if you omit specifying an interval qualifier. If you try to set precision on the INTERVAL keyword and include an interval qualifier, Vertica returns an error.
- You can set precision only on the last unit of an interval qualifier. For example:

```
CREATE TABLE public.testint2
(
  i INTERVAL HOUR TO SECOND(3)
);
```

If you specify precision on another unit, Vertica discards it when it saves the table definition.

Fractional Seconds in Interval Units

Vertica supports intervals in milliseconds (hh:mm:ss:ms), where 01:02:03:25 represents 1 hour, 2 minutes, 3 seconds, and 025 milliseconds. Milliseconds are converted to fractional seconds as in the following example, which returns 1 day, 2 hours, 3 minutes, 4 seconds, and 25.5 milliseconds:

```
=> SELECT INTERVAL '1 02:03:04:25.5';
?column?
-----
1 day 02:03:04.0255
```

Vertica allows fractional minutes. The fractional minutes are rounded into seconds:

```
=> SELECT INTERVAL '10.5 minutes';
?column?
-----
00:10:30
=> select interval '10.659 minutes';
?column?
-----
00:10:39.54
=> select interval '10.3333333333333333 minutes';
?column?
-----
00:10:20
```

Considerations

- An INTERVAL can include only the subset of units that you need; however, year/month intervals represent calendar years and months with no fixed number of days, so year/month interval values cannot include days, hours, minutes. When year/month values are specified for day/time intervals, the intervals extension assumes 30 days per month and 365 days per

year. Since the length of a given month or year varies, day/time intervals are never output as months or years, only as days, hours, minutes, and so on.

- Day/time and year/month intervals are logically independent and cannot be combined with or compared to each other. In the following example, an interval-literal that contains DAYS cannot be combined with the YEAR TO MONTH type:

```
=> SELECT INTERVAL '1 2 3' YEAR TO MONTH;
ERROR 3679: Invalid input syntax for interval year to month: "1 2 3"
```

- Vertica accepts intervals up to $2^{63} - 1$ microseconds or months (about 18 digits).
- INTERVAL YEAR TO MONTH can be used in an analytic [RANGE window](#) when the ORDER BY column type is `TIMESTAMP/TIMESTAMP WITH TIMEZONE`, or `DATE`. Using `TIME/TIME WITH TIMEZONE` are not supported.
- You can use INTERVAL DAY TO SECOND when the ORDER BY column type is `TIMESTAMP/TIMESTAMP WITH TIMEZONE`, `DATE`, and `TIME/TIME WITH TIMEZONE`.

Examples

Examples in this section assume that INTERVALSTYLE is set to PLAIN , so results omit [subtype units](#). Interval values that omit an [interval qualifier](#) use the default to DAY TO SECOND(6).

Query	Result
SELECT INTERVAL '00:2500:00';	1 17:40
SELECT INTERVAL '2500' MINUTE TO SECOND;	2500
SELECT INTERVAL '2500' MINUTE;	2500
SELECT INTERVAL '28 days 3 hours' HOUR TO SECOND;	675:00
SELECT INTERVAL(3) '28 days 3 hours';	28 03:00
SELECT INTERVAL(3) '28 days 3 hours 1.234567';	28 03:01:14.074
SELECT INTERVAL(3) '28 days 3 hours 1.234567 sec';	28 03:00:01.235
SELECT INTERVAL(3) '28 days 3.3 hours' HOUR TO SECOND;	675:18
SELECT INTERVAL(3) '28 days 3.35 hours' HOUR TO SECOND;	675:21
SELECT INTERVAL(3) '28 days 3.37 hours' HOUR TO SECOND;	675:22:12
SELECT INTERVAL '1.234567 days' HOUR TO SECOND;	29:37:46.5888
SELECT INTERVAL '1.23456789 days' HOUR TO SECOND;	29:37:46.665696

Query	Result
SELECT INTERVAL(3) '1.23456789 days' HOUR TO SECOND;	29:37:46.666
SELECT INTERVAL(3) '1.23456789 days' HOUR TO SECOND(2);	29:37:46.67
SELECT INTERVAL(3) '01:00:01.234567' as "one hour+";	01:00:01.235
SELECT INTERVAL(3) '01:00:01.234567' = INTERVAL(3) '01:00:01.234567';	t
SELECT INTERVAL(3) '01:00:01.234567' = INTERVAL '01:00:01.234567';	f
SELECT INTERVAL(3) '01:00:01.234567' = INTERVAL '01:00:01.234567' HOUR TO SECOND(3);	t
SELECT INTERVAL(3) '01:00:01.234567' = INTERVAL '01:00:01.234567' MINUTE TO SECOND (3);	t
SELECT INTERVAL '255 1.1111' MINUTE TO SECOND(3);	255:01.111
SELECT INTERVAL '@ - 5 ago';	5
SELECT INTERVAL '@ - 5 minutes ago';	00:05
SELECT INTERVAL '@ 5 minutes ago';	-00:05
SELECT INTERVAL '@ ago -5 minutes';	00:05
SELECT DATE_PART('month', INTERVAL '2-3' YEAR TO MONTH);	3
SELECT FLOOR((TIMESTAMP '2005-01-17 10:00' - TIMESTAMP '2005-01-01') / INTERVAL '7');	2

Processing Signed Intervals

In the SQL:2008 standard, a minus sign before an interval-literal or as the first character of the interval-literal negates the entire literal, not just the first component. In Vertica, a leading minus sign negates the entire interval, not just the first component. The following commands both return the same value:

```
=> SELECT INTERVAL '-1 month - 1 second';
?column?
-----
-29 days 23:59:59

=> SELECT INTERVAL -'1 month - 1 second';
?column?
-----
-29 days 23:59:59
```

Use one of the following commands instead to return the intended result:

```
=> SELECT INTERVAL -'1 month 1 second';
?column?
```

```
-----  
-30 days 1 sec  
=> SELECT INTERVAL '-30 00:00:01';  
?column?  
-----  
-30 days 1 sec
```

Two negatives together return a positive:

```
=> SELECT INTERVAL '--1 month - 1 second';  
?column?  
-----  
29 days 23:59:59  
=> SELECT INTERVAL '--1 month 1 second';  
?column?  
-----  
30 days 1 sec
```

You can use the year-month syntax with no spaces. Vertica allows the input of negative months but requires two negatives when paired with years.

```
=> SELECT INTERVAL '3-3' YEAR TO MONTH;  
?column?  
-----  
3 years 3 months  
=> SELECT INTERVAL '3--3' YEAR TO MONTH;  
?column?  
-----  
2 years 9 months
```

When the interval-literal looks like a year/month type, but the type is day/second, or vice versa, Vertica reads the interval-literal from left to right, where number-number is years-months, and number <space> <signed number> is whatever the units specify. Vertica processes the following command as $(-)$ 1 year 1 month = $(-)$ 365 + 30 = -395 days:

```
=> SELECT INTERVAL '-1-1' DAY TO HOUR;  
?column?  
-----  
-395 days
```

If you insert a space in the interval-literal, Vertica processes it based on the subtype DAY TO HOUR: $(-)$ 1 day – 1 hour = $(-)$ 24 – 1 = -23 hours:

```
=> SELECT INTERVAL '-1 -1' DAY TO HOUR;  
?column?  
-----  
-23 hours
```

Two negatives together returns a positive, so Vertica processes the following command as $(-)$ 1 year – 1 month = $(-)$ 365 – 30 = -335 days:

```
=> SELECT INTERVAL '-1--1' DAY TO HOUR;
?column?
-----
-335 days
```

If you omit the value after the hyphen, Vertica assumes 0 months and processes the following command as 1 year 0 month –1 day = 365 + 0 – 1 = –364 days:

```
=> SELECT INTERVAL '1- -1' DAY TO HOUR;
?column?
-----
364 days
```

Casting with Intervals

You can use CAST to convert strings to intervals, and vice versa.

String to Interval

You cast a string to an interval as follows:

```
CAST( [ INTERVAL[(p)] ] [-] ] interval-Literal AS INTERVAL[(p)] interval-qualifier )
```

For example:

```
=> SELECT CAST('3700 sec' AS INTERVAL);
?column?
-----
01:01:40
```

You can cast intervals within day-time or the year-month subtypes but not between them:

```
=> SELECT CAST(INTERVAL '4440' MINUTE as INTERVAL);
?column?
-----
3 days 2 hours
=> SELECT CAST(INTERVAL -'01:15' as INTERVAL MINUTE);
?column?
-----
-75 mins
```

Interval to String

You cast an interval to a string as follows:

```
CAST( (SELECT interval ) AS VARCHAR[(n)] )
```

For example:

```
=> SELECT CONCAT(
  'Tomorrow at this time: ',
  CAST((SELECT INTERVAL '24 hours') + CURRENT_TIMESTAMP(0) AS VARCHAR));
      CONCAT
-----
Tomorrow at this time: 2016-08-17 08:41:23-04
(1 row)
```

Operations with Intervals

If you divide an interval by an interval, you get a FLOAT:

```
=> SELECT INTERVAL '28 days 3 hours' HOUR(4) / INTERVAL '27 days 3 hours' HOUR(4);
      ?column?
-----
1.036866359447
```

An INTERVAL divided by FLOAT returns an INTERVAL:

```
=> SELECT INTERVAL '3' MINUTE / 1.5;
      ?column?
-----
2 mins
```

INTERVAL MODULO (remainder) INTERVAL returns an INTERVAL:

```
=> SELECT INTERVAL '28 days 3 hours' HOUR % INTERVAL '27 days 3 hours' HOUR;
      ?column?
-----
24 hours
```

If you add INTERVAL and TIME, the result is TIME, modulo 24 hours:

```
=> SELECT INTERVAL '1' HOUR + TIME '1:30';
      ?column?
-----
02:30:00
```

SMALLDATETIME

SMALLDATETIME is an alias for [TIMESTAMP/TIMESTAMPTZ](#).

TIME/TIMETZ

Stores the specified time of day. TIMETZ is the same as TIME WITH TIME ZONE: both data types store the UTC offset of the specified time.

Syntax

TIME

```
TIME [ (p) ] [ { WITHOUT | WITH } TIME ZONE ] 'input-string' [ AT TIME ZONE zone ]
```

TIMETZ

```
TIME [ (p) ] 'input-string' [ AT TIME ZONE zone ]
```

Parameters

<i>p</i>	Optional precision value that specifies the number of fractional digits retained in the seconds field, an integer value between 0 and 6. If you omit specifying precision, Vertica returns up to 6 fractional digits.
WITHOUT TIME ZONE WITH TIME ZONE	Specifies whether to include a time zone with the stored value: <ul style="list-style-type: none"> WITHOUT TIME ZONE (default): Specifies that <i>input-string</i> does not include a time zone. If the input string contains a time zone, Vertica ignores this qualifier. Instead, it conforms to WITH TIME ZONE behavior. WITH TIME ZONE: Specifies to convert <i>input-string</i> to UTC, using the UTC offset for the specified time zone. If the input string omits a time zone, Vertica uses the UTC offset of the time zone that is configured for your system.
<i>input-string</i>	See Input String below.
AT TIME ZONE <i>zone</i>	See TIME AT TIME ZONE and TIMESTAMP AT TIME ZONE .

TIME versus TIMETZ

TIMETZ and [TIMESTAMPTZ](#) are not parallel SQL constructs. TIMESTAMPTZ records a time and date in GMT, converting from the specified TIME_ZONE. TIMETZ records the specified time and the specified time zone, in minutes, from GMT.

Limits

Name	Low Value	High Value	Resolution
TIME [p]	00:00:00.00	23:59:60.999999	1 μ s
TIME [p] WITH TIME ZONE	00:00:00.00+14	23:59:59.999999-14	1 μ s

Input String

A TIME input string can be set to any of the formats shown below:

Example	Description
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	Same as 04:05; AM does not affect value
04:05 PM	Same as 16:05
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	Time zone specified by name

Data Type Coercion

You can cast a TIME or TIMETZ interval to a TIMESTAMP. This returns the local date and time as follows:

```
=> SELECT (TIME '3:01am')::TIMESTAMP;
      ?column?
-----
2012-08-30 03:01:00
(1 row)

=> SELECT (TIMETZ '3:01am')::TIMESTAMP;
      ?column?
-----
2012-08-22 03:01:00
(1 row)
```

Casting the same TIME or TIMETZ interval to a TIMESTAMPTZ returns the local date and time, appended with the UTC offset—in this example, -05:

```
=> SELECT (TIME '3:01am')::TIMESTAMPTZ;
      ?column?
-----
2016-12-08 03:01:00-05
(1 row)
```

TIME AT TIME ZONE

Converts the specified TIME to the time in another time zone.

Syntax

```
TIME [WITH TIME ZONE] 'input-string' AT TIME ZONE 'zone'
```

Parameters

<code>WITH TIME ZONE</code>	Converts the input string to UTC, using the UTC offset for the specified time zone. If the input string omits a time zone, Vertica uses the UTC offset of the time zone that is configured for your system , and converts the input string accordingly
<code>zone</code>	Specifies the time zone to use in the conversion, either as a literal or

interval that specifies UTC offset:

- AT TIME ZONE INTERVAL '*utc-offset*'
- AT TIME ZONE '*time-zone-literal*'

For details, see [Specifying Time Zones](#) below.

Note: Vertica treats literals TIME ZONE and TIMEZONE as synonyms.

Specifying Time Zones

You can specify time zones in two ways:

- A string literal such as America/Chicago or PST
- An interval that specifies a UTC offset—for example, INTERVAL '-08:00'

It is generally good practice to specify time zones with literals that indicate a geographic location. Vertica makes the necessary seasonal adjustments, and thereby avoids inconsistent results. For example, the following two queries are issued when daylight time is in effect. Because the local UTC offset during daylight time is -04, both queries return the same results:

```
=> SELECT CURRENT_TIME(0) "EDT";
      EDT
-----
12:34:35-04
(1 row)

=> SELECT CURRENT_TIME(0) AT TIME ZONE 'America/Denver' "Mountain Time";
      Mountain Time
-----
10:34:35-06
(1 row)

=> SELECT CURRENT_TIME(0) AT TIME ZONE INTERVAL '-06:00' "Mountain Time";
      Mountain Time
-----
10:34:35-06
(1 row)
```

If you use the UTC offset in a similar query when standard time is in effect, you must adjust the UTC offset accordingly—for Denver time, to -07—otherwise, Vertica returns a different (and erroneous) result:

```
=> SELECT CURRENT_TIME(0) "EST";
      EST
-----
14:18:22-05
(1 row)

=> SELECT CURRENT_TIME(0) AT TIME ZONE INTERVAL '-06:00' "Mountain Time";
      Mountain Time
-----
13:18:22-06
(1 row)
```

You can show and set the session's time zone with [SHOW TIMEZONE](#) and [SET TIME ZONE](#), respectively:

```
=> SHOW TIMEZONE;
      name |      setting
-----+-----
timezone | America/New_York
(1 row)

=> SELECT CURRENT_TIME(0) "Eastern Daylight Time";
      Eastern Daylight Time
-----
12:18:24-04
(1 row)

=> SET TIMEZONE 'America/Los_Angeles';
SET

=> SELECT CURRENT_TIME(0) "Pacific Daylight Time";
      Pacific Daylight Time
-----
09:18:24-07
(1 row)
```

Time Zone Literals

To view the default list of valid literals, see the files in the following directory:

opt/vertica/share/timezonesets

For example:

```
$ cat Antarctica.txt
...
# src/timezone/tznames/Antarctica.txt
#
AWST    28800    # Australian Western Standard Time
          #      (Antarctica/Casey)
          #      (Australia/Perth)
...
```

```
NZST    43200    # New Zealand Standard Time
          #      (Antarctica/McMurdo)
          #      (Pacific/Auckland)
ROTT    -10800    # Rothera Time
          #      (Antarctica/Rothera)
SYOT    10800    # Syowa Time
          #      (Antarctica/Syowa)
VOST    21600    # Vostok time
          #      (Antarctica/Vostok)
```

Examples

The following example assumes that local time is EST (Eastern Standard Time). The query converts the specified time to MST (mountain standard time):

```
=> SELECT CURRENT_TIME(0);
      timezone
-----
10:10:56-05
(1 row)

=> SELECT TIME '10:10:56' AT TIME ZONE 'America/Denver' "Denver Time";
      Denver Time
-----
08:10:56-07
(1 row)
```

The next example adds a time zone literal to the input string—in this case, Europe/Vilnius—and converts the time to MST:

```
=> SELECT TIME '09:56:13 Europe/Vilnius' AT TIME ZONE 'America/Denver';
      Denver Time
-----
00:56:13-07
(1 row)
```

TIMESTAMP/TIMESTAMPTZ

Stores the specified date and time. `TIMESTAMPTZ` is the same as `TIMESTAMP WITH TIME ZONE`: both data types store the UTC offset of the specified time.

`TIMESTAMP` is an alias for `DATETIME` and `SMALLDATETIME`.

Syntax

TIMESTAMP

```
TIMESTAMP [ (p) ] [ { WITHOUT | WITH } TIME ZONE ] 'input-string' [ AT TIME ZONE zone ]
```

TIMESTAMPTZ

```
TIMESTAMPTZ [ (p) ] 'input-string' [ AT TIME ZONE zone ]
```

Parameters

<i>p</i>	Optional precision value that specifies the number of fractional digits retained in the seconds field, an integer value between 0 and 6. If you omit specifying precision, Vertica returns up to 6 fractional digits.
WITHOUT TIME ZONE WITH TIME ZONE	Specifies whether to include a time zone with the stored value: <ul style="list-style-type: none"> WITHOUT TIME ZONE (default): Specifies that <i>input-string</i> does not include a time zone. If the input string contains a time zone, Vertica ignores this qualifier. Instead, it conforms to WITH TIME ZONE behavior. WITH TIME ZONE: Specifies to convert <i>input-string</i> to UTC, using the UTC offset for the specified time zone. If the input string omits a time zone, Vertica uses the UTC offset of the time zone that is configured for your system.
<i>input-string</i>	See Input String below.
AT TIME ZONE <i>zone</i>	See TIMESTAMP AT TIME ZONE .

Limits

In the following table, values are rounded. See [Date/Time Data Types](#) for more detail.

Name	Low Value	High Value	Resolution
TIMESTAMP [(p)] [WITHOUT TIME ZONE]	290279 BC	294277 AD	1 μ s
TIMESTAMP [(p)] WITH TIME ZONE	290279 BC	294277 AD	1 μ s

Input String

The date/time input string concatenates a date and a time. The input string can include a time zone, specified as a literal such as *America/Chicago*, or as a UTC offset.

The following list represents typical date/time input variations:

- 1999-01-08 04:05:06
- 1999-01-08 04:05:06 -8:00
- January 8 04:05:06 1999 PST

The input string can also specify the calendar era, either AD (default) or BC. If you omit the calendar era, Vertica assumes the current calendar era (AD). The calendar era typically follows the time zone; however, the input string can include it in various locations. For example, the following queries return the same results:

```
=> SELECT TIMESTAMP WITH TIME ZONE 'March 1, 44 12:00 CET BC ' "Caesar's Time of Death EST";
Caesar's Time of Death EST
-----
0044-03-01 06:00:00-05 BC
(1 row)

=> SELECT TIMESTAMP WITH TIME ZONE 'March 1, 44 12:00 BC CET' "Caesar's Time of Death EST";
Caesar's Time of Death EST
-----
0044-03-01 06:00:00-05 BC
(1 row)
```

Examples: TIMESTAMP Computation

Statement	Returns
SELECT (TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01');	16 10:10
SELECT (TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') / 7;	2 08:17:08.571429
SELECT TIMESTAMP '2009-05-29 15:21:00.456789'-TIMESTAMP '2009-05-28';	1 15:21:00.456789
SELECT (TIMESTAMP '2009-05-29 15:21:00.456789'-TIMESTAMP '2009-05-28')(3);	1 15:21:00.457
SELECT '2017-03-18 07:00'::TIMESTAMPTZ(0) + INTERVAL '1.5 day';	2017-03-19 19:00:00-04
SELECT (TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') day;	16
SELECT cast((TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') day as integer) / 7;	2
SELECT floor((TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') / interval '7');	2
SELECT (TIMESTAMP '2009-05-29 15:21:00.456789'-TIMESTAMP '2009-05-28')second;	141660.456789
SELECT (TIMESTAMP '2012-05-29 15:21:00.456789'-TIMESTAMP '2009-01-01') year;	3
SELECT (TIMESTAMP '2012-05-29 15:21:00.456789'-TIMESTAMP '2009-01-01') month;	40
SELECT (TIMESTAMP '2012-05-29 15:21:00.456789'-TIMESTAMP '2009-01-01') year to month;	3-4
SELECT (TIMESTAMP '2012-05-29 15:21:00.456789'-TIMESTAMP '2009-01-01') second (3);	107536860.457
SELECT (TIMESTAMP '2012-05-29 15:21:00.456789'-TIMESTAMP '2009-01-01') minute;	1792281
SELECT (TIMESTAMP '2012-05-29 15:21:00.456789'-TIMESTAMP '2009-01-01') minute to second(3);	1792281:00.457
SELECT TIMESTAMP 'infinity';	infinity

TIMESTAMP AT TIME ZONE

Converts the specified `TIMESTAMP` or `TIMESTAMPTZ` (`TIMESTAMP WITH TIMEZONE`) to another time zone. Vertica executes `AT TIME ZONE` differently, depending on whether the date input is a `TIMESTAMP` or `TIMESTAMPTZ`. See [TIMESTAMP Versus TIMESTAMPTZ Behavior](#) below.

Syntax

timestamp-clause AT TIME ZONE 'zone'

Parameters

<i>timestamp-clause</i>	<p>Specifies the timestamp to convert, either <code>TIMESTAMP</code> or <code>TIMESTAMPTZ</code>.</p> <p>For details, see TIMESTAMP/TIMESTAMPTZ.</p>
AT TIME ZONE <i>zone</i>	<p>Specifies the time zone to use in the timestamp conversion, where <i>zone</i> is a literal or interval that specifies a UTC offset:</p> <ul style="list-style-type: none"> • AT TIME ZONE INTERVAL '<i>utc-offset</i>' • AT TIME ZONE '<i>time-zone-literal</i>' <p>For details, see Specifying Time Zones below.</p> <p>Note: Vertica treats literals <code>TIME ZONE</code> and <code>TIMEZONE</code> as synonyms.</p>

TIMESTAMP Versus TIMESTAMPTZ Behavior

How Vertica interprets `AT TIME ZONE` depends on whether the date input is a `TIMESTAMP` or `TIMESTAMPTZ`:

Date input	Action
<code>TIMESTAMP</code>	<p>If the input string specifies no time zone, Vertica performs two actions:</p> <ol style="list-style-type: none"> 1. Converts the input string to the time zone of the <code>AT TIME ZONE</code> argument. 2. Returns the time for the current session's time zone. <p>If the input string includes a time zone, Vertica implicitly casts it to a <code>TIMESTAMPTZ</code> and converts it accordingly (see <code>TIMESTAMPTZ</code> below).</p> <p>For example, the following statement specifies a <code>TIMESTAMP</code> with no time</p>

Date input	Action
	<p>zone. Vertica executes the statement as follows:</p> <ol style="list-style-type: none"> 1. Converts the input string to PDT (Pacific Daylight Time). 2. Returns that time in the local time zone, which is three hours later: <pre data-bbox="431 495 1398 831"> => SHOW TIMEZONE; name setting -----+----- timezone America/New_York (1 row) SELECT TIMESTAMP '2017-3-14 5:30' AT TIME ZONE 'PDT'; timezone ----- 2017-03-14 08:30:00-04 (1 row) </pre>
TIMESTAMPTZ	<p>Vertica converts the input string to the time zone of the AT TIME ZONE argument and returns that time.</p> <p>For example, the following statement specifies a TIMESTAMPTZ data type. The input string omits any time zone expression, so Vertica assumes the input string to be in local time zone (America/New_York) and returns the time of the AT TIME ZONE argument:</p> <pre data-bbox="431 1146 1398 1503"> => SHOW TIMEZONE; name setting -----+----- timezone America/New_York (1 row) => SELECT TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40' AT TIME ZONE 'America/Denver'; timezone ----- 2001-02-16 18:38:40 (1 row) </pre> <p>The input string in the next statement explicitly specifies a time zone, so Vertica coerces the TIMESTAMP to a TIMESTAMPTZ and returns the time of the AT TIME ZONE argument:</p> <pre data-bbox="431 1671 1398 1860"> => SELECT TIMESTAMP '2001-02-16 20:38:40 America/Mexico_City' AT TIME ZONE 'Asia/Tokyo'; timezone ----- 2001-02-17 11:38:40 (1 row) </pre>

Specifying Time Zones

You can specify time zones in two ways:

- A string literal such as `America/Chicago` or `PST`
- An interval that specifies a UTC offset—for example, `INTERVAL '-08:00'`

It is generally good practice to specify time zones with literals that indicate a geographic location. Vertica makes the necessary seasonal adjustments, and thereby avoids inconsistent results. For example, the following two queries are issued when daylight time is in effect. Because the local UTC offset during daylight time is `-04`, both queries return the same results:

```
=> SELECT TIMESTAMPTZ '2017-03-16 09:56:13' AT TIME ZONE 'America/Denver' "Denver Time";
       Denver Time
-----
2017-03-16 07:56:13
(1 row)

=> SELECT TIMESTAMPTZ '2017-03-16 09:56:13' AT TIME ZONE INTERVAL '-06:00' "Denver Time";
       Denver Time
-----
2017-03-16 07:56:13
(1 row)
```

If you issue a use the UTC offset in a similar query when standard time is in effect, you must adjust the UTC offset accordingly—for Denver time, to `-07`—otherwise, Vertica returns a different (and erroneous) result:

```
=> SELECT TIMESTAMPTZ '2017-01-16 09:56:13' AT TIME ZONE 'America/Denver' "Denver Time";
       Denver Time
-----
2017-01-16 07:56:13
(1 row)

=> SELECT TIMESTAMPTZ '2017-01-16 09:56:13' AT TIME ZONE INTERVAL '-06:00' "Denver Time";
       Denver Time
-----
2017-01-16 08:56:13
(1 row)
```

You can show and set the session's time zone with [SHOW TIMEZONE](#) and [SET TIME ZONE](#), respectively:

```
=> SHOW TIMEZONE;
 name | setting
-----+-----
timezone | America/New_York
(1 row)
```

```
=> SELECT CURRENT_TIMESTAMP(0) "Eastern Daylight Time";
Eastern Daylight Time
-----
2017-03-20 12:18:24-04
(1 row)

=> SET TIMEZONE 'America/Los_Angeles';
SET

=> SELECT CURRENT_TIMESTAMP(0) "Pacific Daylight Time";
Pacific Daylight Time
-----
2017-03-20 09:18:24-07
(1 row)
```

Time Zone Literals

To view the default list of valid literals, see the files in the following directory:

`opt/vertica/share/timezonesets`

For example:

```
$ cat Antarctica.txt
...
# src/timezone/tznames/Antarctica.txt
#

AWST    28800    # Australian Western Standard Time
        #      (Antarctica/Casey)
        #      (Australia/Perth)
...

NZST    43200    # New Zealand Standard Time
        #      (Antarctica/McMurdo)
        #      (Pacific/Auckland)

ROTT    -10800    # Rothera Time
        #      (Antarctica/Rothera)

SYOT    10800    # Syowa Time
        #      (Antarctica/Syowa)

VOST    21600    # Vostok time
        #      (Antarctica/Vostok)
```

Examples

The following examples assume that the current time zone is `America/New_York`:

Convert the string `10:00` to the specified time zone:

```
=> SELECT TIME '10:00' AT TIME ZONE 'America/Chicago';
timezone
-----
```

```
09:00:00-06  
(1 row)
```

Long Data Types

Store data up to 32,000,000 bytes:

- `LONG VARBINARY`—Variable-length raw-byte data, such as spatial data. `LONG VARBINARY` values are not extended to the full width of the column.
- `LONG VARCHAR`—Variable-length strings, such as log files and unstructured data. `LONG VARCHAR` values are not extended to the full width of the column.

The maximum size for the `LONG` data types is 32,000,000 bytes. Use the `LONG` data types only when you need to store data greater than 65,000 bytes, which is the maximum size for `VARBINARY` and `VARCHAR` data types. Such data might include unstructured data, online comments or posts, or small log files.

Default length is 1048576.

Flex tables have a default `LONG VARBINARY __raw__` column, with a `NOT NULL` constraint. For more information, see [Using Flex Tables](#).

Syntax

```
LONG VARBINARY (max_Length )  
LONG VARCHAR (octet_Length )
```

Parameters

<i>max_Length</i>	Specifies the length of the byte string (column width, declared in bytes (octets)), in CREATE TABLE statements). Maximum value: 32,000,000.
<i>octet_Length</i>	Specifies the length of the string (column width, declared in bytes (octets)), in CREATE TABLE statements). Maximum value: 32,000,000.

Notes

For optimal performance of LONG data types, Vertica recommends that you:

- Use the LONG data types as *storage only* containers; Vertica supports operations on the content of LONG data types, but not all the operations that VARCHAR and VARBINARY take.
- Use the VARBINARY and VARCHAR data types, instead of their LONG counterparts, whenever possible. The VARBINARY and VARCHAR data types are more flexible and have a wider range of operations.
- Do not sort, segment, or partition projections on LONG data type columns.
- Do not add constraints, such as a primary key, to any LONG VARBINARY or LONG VARCHAR columns.
- Do not join or aggregate any LONG data type columns.

Example

The following example creates a table `user_comments` with a LONG VARCHAR column and inserts data into it:

```
=> CREATE TABLE user_comments
      (id INTEGER,
       username VARCHAR(200),
       time_posted TIMESTAMP,
       comment_text LONG VARCHAR(200000));
=> INSERT INTO user_comments VALUES
      (1,
       'User1',
       TIMESTAMP '2013-06-25 12:47:32.62',
       'The weather tomorrow will be cold and rainy and then
       on the day after, the sun will come and the temperature
       will rise dramatically.');
```

Numeric Data Types

Numeric data types are numbers stored in database columns. These data types are typically grouped by:

- **Exact** numeric types, values where the precision and scale need to be preserved. The exact numeric types are `INTEGER`, `BIGINT`, `DECIMAL`, `NUMERIC`, `NUMBER`, and `MONEY`.
- **Approximate** numeric types, values where the precision needs to be preserved and the scale can be floating. The approximate numeric types are `DOUBLE PRECISION`, `FLOAT`, and `REAL`.

Implicit casts from `INTEGER`, `FLOAT`, and `NUMERIC` to `VARCHAR` are not supported. If you need that functionality, write an explicit cast using one of the following forms:

```
CAST(x AS data-type-name) or x::data-type-name
```

The following example casts a float to an integer:

```
=> SELECT(FLOAT '123.5')::INT;
?column?
-----
      124
(1 row)
```

String-to-numeric data type conversions accept formats of quoted constants for scientific notation, binary scaling, hexadecimal, and combinations of numeric-type literals:

- **Scientific notation :**

```
=> SELECT FLOAT '1e10';
?column?
-----
10000000000
(1 row)
```

- **BINARY scaling :**

```
=> SELECT NUMERIC '1p10';
?column?
-----
      1024
(1 row)
```

- **Hexadecimal:**

```
=> SELECT NUMERIC '0x0abc';  
?column?  
-----  
2748  
(1 row)
```

DOUBLE PRECISION (FLOAT)

Vertica supports the numeric data type `DOUBLE PRECISION`, which is the IEEE-754 8-byte floating point type, along with most of the usual floating point operations.

Syntax

```
[ DOUBLE PRECISION | FLOAT | FLOAT(n) | FLOAT8 | REAL ]
```

Parameters

Note: On a machine whose floating-point arithmetic does not follow IEEE-754, these values probably do not work as expected.

Double precision is an inexact, variable-precision numeric type. In other words, some values cannot be represented exactly and are stored as approximations. Thus, input and output operations involving double precision might show slight discrepancies.

- All of the `DOUBLE PRECISION` data types are synonyms for 64-bit IEEE `FLOAT`.
- The n in `FLOAT(n)` must be between 1 and 53, inclusive, but a 53-bit fraction is always used. See the IEEE-754 standard for details.
- For exact numeric storage and calculations (money for example), use `NUMERIC`.
- Floating point calculations depend on the behavior of the underlying processor, operating system, and compiler.
- Comparing two floating-point values for equality might not work as expected.

Values

`COPY` accepts floating-point data in the following format:

- Optional leading white space
- An optional plus ("+") or minus sign ("-")
- A decimal number, a hexadecimal number, an infinity, a NAN, or a null value

Decimal Number

A decimal number consists of a non-empty sequence of decimal digits possibly containing a radix character (decimal point "."), optionally followed by a decimal exponent. A decimal exponent consists of an "E" or "e", followed by an optional plus or minus sign, followed by a non-empty sequence of decimal digits, and indicates multiplication by a power of 10.

Hexadecimal Number

A hexadecimal number consists of a "0x" or "0X" followed by a non-empty sequence of hexadecimal digits possibly containing a radix character, optionally followed by a binary exponent. A binary exponent consists of a "P" or "p", followed by an optional plus or minus sign, followed by a non-empty sequence of decimal digits, and indicates multiplication by a power of 2. At least one of radix character and binary exponent must be present.

Infinity

An infinity is either INF or INFINITY, disregarding case.

NaN (Not A Number)

A NaN is NAN (disregarding case) optionally followed by a sequence of characters enclosed in parentheses. The character string specifies the value of NAN in an implementation-dependent manner. (The Vertica internal representation of NAN is 0xfff8000000000000LL on x86 machines.)

When writing infinity or NAN values as constants in a SQL statement, enclose them in single quotes. For example:

```
=> UPDATE table SET x = 'Infinity'
```

Note: Vertica follows the IEEE definition of NaNs (IEEE 754). The SQL standards do not specify how floating point works in detail.

IEEE defines NaNs as a set of floating point values where each one is not equal to anything, even to itself. A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved.

However, for the purpose of sorting data, NaN values must be placed somewhere in the result. The value generated 'NaN' appears in the context of a floating point number

matches the NaN value generated by the hardware. For example, Intel hardware generates (0xffff800000000000LL), which is technically a Negative, Quiet, Non-signaling NaN.

Vertica uses a different NaN value to represent floating point NULL (0x7ffffffffffffeLL). This is a Positive, Quiet, Non-signaling NaN and is reserved by Vertica

A NaN example follows.

```
=> SELECT CBRT('NaN'); -- cube root
      CBRT
-----
      NaN
(1 row)

=> SELECT 'NaN' > 1.0;
      ?column?
-----
           f
(1 row)
```

Null Value

The load file format of a null value is user defined, as described in the COPY command. The Vertica internal representation of a null value is 0x7ffffffffffffeLL. The interactive format is controlled by the vsql printing option null. For example:

```
\pset null '(null)'
```

The default option is not to print anything.

Rules

- $-0 == +0$
- $1/0 = \text{Infinity}$
- $0/0 == \text{Nan}$
- $\text{NaN} \neq \text{anything}$ (even NaN)

To search for NaN column values, use the following predicate:

```
... WHERE column != column
```

This is necessary because WHERE *column* = 'NaN' cannot be true by definition.

Sort Order (Ascending)

- NaN
- -Inf
- numbers
- +Inf
- NULL

Notes

- NULL appears last (largest) in ascending order.
- All overflows in floats generate +/-infinity or NaN, per the IEEE floating point standard.

INTEGER

A signed 8-byte (64-bit) data type.

Syntax

```
[ INTEGER | INT | BIGINT | INT8 | SMALLINT | TINYINT ]
```

Parameters

INT, INTEGER, INT8, SMALLINT, TINYINT, and BIGINT are all synonyms for the same signed 64-bit integer data type. Automatic compression techniques are used to conserve disk space in cases where the full 64 bits are not required.

Notes

- The range of values is $-2^{63}+1$ to $2^{63}-1$.
- $2^{63} = 9,223,372,036,854,775,808$ (19 digits).

- The value -2^{63} is reserved to represent NULL.
- NULL appears first (smallest) in ascending order.
- Vertica does not have an explicit 4-byte (32-bit integer) or smaller types. Vertica's encoding and compression automatically eliminate the storage overhead of values that fit in less than 64 bits.

Restrictions

- The JDBC type INTEGER is 4 bytes and is not supported by Vertica. Use BIGINT instead.
- Vertica does not support the SQL/JDBC types NUMERIC, SMALLINT, or TINYINT.
- Vertica does not check for overflow (positive or negative) except in the aggregate function SUM(). If you encounter overflow when using SUM, use SUM_FLOAT(), which converts to floating point.

See Also

[Data Type Coercion Chart](#)

NUMERIC

Numeric data types store fixed point numeric data. For example, a money value of \$123.45 can be stored in a NUMERIC(5, 2) field.

Syntax

NUMERIC | DECIMAL | NUMBER | MONEY [(*precision* [, *scale*])]

Parameters

<i>precision</i>	The total number of significant digits that the data type stores. <i>precision</i> must be positive and ≤ 1024 . If you assign a value that exceeds the <i>precision</i> value, an error occurs.
------------------	---

<i>scale</i>	The maximum number of digits to the right of the decimal point that the data type stores. <i>scale</i> must be non-negative and less than or equal to <i>precision</i> . If you omit the <i>scale</i> parameter, the <i>scale</i> value is set to 0. If you assign a value with more decimal digits than <i>scale</i> , the value is rounded to <i>scale</i> digits.
--------------	--

Notes

- The maximum number of significant digits (digits to the left of the decimal point) is equal to precision minus scale. For example, NUMERIC(9, 6) allows three significant digits. In this case, if you insert a value that has more than three significant digits, an error occurs.
- Numeric data types that have a precision of 18 or less have similar performance characteristics as INTEGER, regardless of the scale. When possible, Vertica recommends using a precision <= 18.
- NUMERIC, DECIMAL, NUMBER, and MONEY are all synonyms that return NUMERIC data. The four numeric data types differ in their default precision and scale values:

Type	Precision	Scale
NUMERIC	37	15
DECIMAL	37	15
NUMBER	38	0
MONEY	18	4

- Numeric data types are *exact* data types that store values of a specified precision and scale, expressed with a number of digits before and after a decimal point. This contrasts with the Vertica integer and floating data types:

The **DOUBLE PRECISION (FLOAT)** type supports ~15 digits, variable exponent, and represents numeric values approximately.

The **INTEGER** type supports ~18 digits, whole numbers only.

- The *approximate* numeric data type, DOUBLE PRECISION, uses floating points and may be less precise than NUMERIC data types.
- Supported numeric operations include the following:

Basic math: +, -, *, /

Aggregation: SUM, MIN, MAX, COUNT

Comparison operators: <, <=, =, <=>, <>, >, >=

- NUMERIC divide operates directly on numeric values, without converting to floating point. The result has at least 18 decimal places and is rounded.
- NUMERIC mod (including %) operates directly on numeric values, without converting to floating point. The result has the same scale as the numerator and never needs rounding.
- NULL appears first (smallest) in ascending order.
- COPY accepts a DECIMAL data type with a decimal point ('.'), prefixed by – or +(optional).
- The NUMERIC data type is preferred for non-integer constants, because it is always exact. For example:

```
=> SELECT 1.1 + 2.2 = 3.3;
?column?
-----
t
(1 row)

=> SELECT 1.1::float + 2.2::float = 3.3::float;
?column?
-----
f
(1 row)
```

- Performance of the NUMERIC data type has been fine tuned for the common case of 18 digits of precision.
- Some of the more complex operations used with numeric data types result in an implicit cast to FLOAT. When using SQRT, STDDEV, transcendental functions such as LOG, and TO_CHAR/TO_NUMBER formatting, the result is always FLOAT.
- For valid encoding types for numeric data type columns, see [Encoding Types](#).

Examples

The following series of commands creates a table that contains a numeric data type and then performs some mathematical operations on the data:

```
=> CREATE TABLE num1 (id INTEGER, amount NUMERIC(8,2));
```

Insert some values into the table:

```
=> INSERT INTO num1 VALUES (1, 123456.78);
```

Query the table:

```
=> SELECT * FROM num1;
 id | amount
-----+-----
   1 | 123456.78
(1 row)
```

The following example returns the NUMERIC column, amount, from table num1:

```
=> SELECT amount FROM num1;
 amount
-----
123456.78
(1 row)
```

The following syntax adds one (1) to the amount:

```
=> SELECT amount+1 AS 'amount' FROM num1;
 amount
-----
123457.78
(1 row)
```

The following syntax multiplies the amount column by 2:

```
=> SELECT amount*2 AS 'amount' FROM num1;
 amount
-----
246913.56
(1 row)
```

The following syntax returns a negative number for the amount column:

```
=> SELECT -amount FROM num1;
?column?
-----
-123456.78
(1 row)
```

The following syntax returns the absolute value of the amount argument:

```
=> SELECT ABS(amount) FROM num1;
 ABS
-----
123456.78
(1 row)
```

The following syntax casts the NUMERIC amount as a FLOAT data type:

```
=> SELECT amount::float FROM num1;
   amount
-----
 123456.78
(1 row)
```

See Also

[Mathematical Functions](#)

[Encoding Types](#)

Numeric Data Type Overflow

Vertica does not check for overflow (positive or negative) except in the aggregate function `SUM()`. If you encounter overflow when using `SUM`, use `SUM_FLOAT()` which converts to floating point.

For a detailed discussion of how Vertica handles overflow when you use the functions `SUM`, `SUM_FLOAT`, and `AVG` with numeric data types, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#). The discussion includes directives for turning off silent numeric overflow and setting precision for numeric data types.

Dividing by zero returns an error:

```
=> SELECT 0/0;
ERROR 3117:  Division by zero

=> SELECT 0.0/0;
ERROR 3117:  Division by zero

=> SELECT 0 // 0;
ERROR 3117:  Division by zero

=> SELECT 200.0/0;
ERROR 3117:  Division by zero

=> SELECT 116.43 // 0;
ERROR 3117:  Division by zero
```

Dividing zero as a `FLOAT` by zero returns `NaN`:

```
=> SELECT 0.0::float/0;
?column?
-----
      NaN
=> SELECT 0.0::float//0;
?column?
```

```
-----  
NaN
```

Dividing a non-zero FLOAT by zero returns Infinity:

```
=> SELECT 2.0::float/0;  
?column?  
-----  
Infinity  
=> SELECT 200.0::float//0;  
?column?  
-----  
Infinity
```

Add, subtract, and multiply operations ignore overflow. Sum and average operations use 128-bit arithmetic internally. `SUM()` reports an error if the final result overflows, suggesting the use of `SUM_FLOAT(INT)`, which converts the 128-bit sum to a `FLOAT8`. For example:

```
=> CREATE TEMP TABLE t (i INT);  
=> INSERT INTO t VALUES (1<<62);  
=> SELECT SUM(i) FROM t;  
ERROR: sum() overflowed  
HINT: try sum_float() instead  
=> SELECT SUM_FLOAT(i) FROM t;  
SUM_FLOAT  
-----  
2.30584300921369e+19
```

Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG

When you use the functions `SUM`, `SUM_FLOAT`, and `AVG` with a `NUMERIC` data type, be aware that overflow can occur and how Vertica responds to that overflow.

This discussion applies to both the aggregate and analytic functions.

For queries, when using the functions `SUM`, `SUM_FLOAT`, and `AVG` with a `NUMERIC` data type, Vertica allows for silent overflow if you exceed your specified precision.

Vertica also allows numeric overflow when you use the `SUM` or `SUM_FLOAT` functions with LAPs.

Default Overflow Handling

With NUMERIC data types, Vertica internally works with multiples of 18 digits. If your specified precision is less than 18 (for example, $x(12, 0)$), Vertica allows for an overflow up to and including the first multiple of 18. In some situations, if you sum a column ($SUM(x)$), you can exceed the number of digits Vertica internally reserves for the result. In this case, Vertica allows a silent overflow.

Turning Off Silent Numeric Overflow

You can turn off silent numeric overflow and instruct Vertica to implicitly include extra digit places. Specifying extra spaces allows Vertica to consistently return your expected results, even when you exceed the precision specified in your DDL.

You turn off silent numeric overflow by setting the parameter `AllowNumericOverflow` to 0 (false).

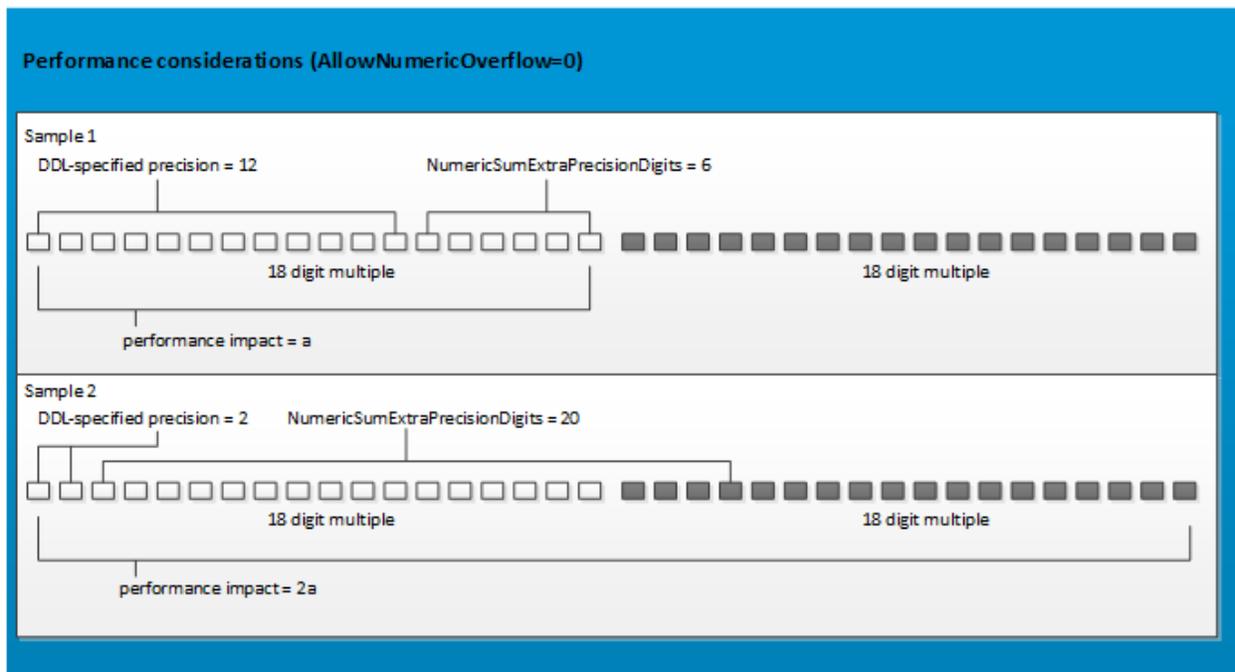
When you set the parameter to 0, Vertica considers the value of a corresponding parameter, `NumericSumExtraPrecisionDigits`.

The `NumericSumExtraPrecisionDigits` parameter defaults to 6, meaning that Vertica internally add six places beyond your DDL-specified precision. Adding extra precision digits can allow Vertica to consistently return results that overflow your DDL-specified precision. However, there can be a performance impact for crossing into the second multiple of 18 internally.

An example:

- Suppose your DDL specifies 11 (for example, $x(11, 0)$) and you accept the default of `NumericSumExtraPrecisionDigits` (6). In this case, Vertica internally stays within the first multiple of 18 digits and no additional performance impact occurs.
- Given the same example, if you set `NumericSumExtraPrecisionDigits` to 10, Vertica internally crosses a threshold into the second multiple of 18. Performance-wise, if (hypothetically) the first example is performance "a," then the second is "2a," substantially increasing the performance impact. Beyond the second multiple of 18, the performance impact continues to be "2a."

This sample representation shows how Vertica responds internally when you set `AllowNumericOverflow` to 0 (false).



Vertica recommends that you turn off silent numeric overflow and set the parameter `NumericSumExtraPrecisionDigits` if you expect to exceed the precision specified in your DDL. Crossing into the second multiple of 18 can affect performance. Therefore, consider carefully before setting `NumericSumExtraPrecisionDigits` to a number higher than what you need for returning the SUM of your numeric columns.

Be aware that, if you turn off `AllowNumericOverflow`, and you exceed the number of extra precision digits set by `NumericSumExtraPrecisionDigits`, Vertica returns an error.

Impact on Live Aggregate Projections (LAPs)

For LAPs, Vertica also allows silent numeric overflow if your LAP uses the `SUM` or `SUM_FLOAT` functions. To turn off silent numeric overflow for LAPs:

1. Set the parameter `AllowNumericOverflow` to 0.
2. Set the parameter `NumericSumExtraPrecisionDigits` to the number of implicit digits you want. Alternatively, use the default setting of 6.
3. Drop and re-create your LAPs.

If you turn off silent numeric overflow, be aware of the following scenarios where an overflow causes a roll back or error message. In these examples, `AllowNumericOverflow` is set to 0 (false), and each LAP uses the `SUM` or `SUM_FLOAT` function.

When numeric overflow is off:

- A load can roll back upon overflow.

Vertica aggregates data before loading in into a LAP. If you are inserting, copying, or merging data, and an overflow occurs during load as Vertica is aggregating the data, Vertica rolls back the load.

- An overflow can occur after load as Vertica sums existing data.

Vertica computes the sum of existing data separately from the computation that it does during data load. If your LAP selects a column using SUM or SUM_FLOAT and an overflow occurs, Vertica produces an error message. This response is similar to the way Vertica produces an error for a query using the SUM or SUM_FLOAT function.

- An overflow can occur during merge-out.

Vertica logs a message during merge-out if an overflow occurs as Vertica computes a final sum during the tuple mover operation. If an error occurs, Vertica marks the LAP as out-of-date. Vertica no longer runs tuple mover operations with the out-of-date LAP.

Data Type Coercion

Vertica supports two types of data type casting:

- *Implicit casting*—Occurs when the expression automatically converts the data from one type to another.
- *Explicit casting*—Occurs when you write a SQL statement that specifies the target data type for the conversion.

The ANSI SQL-92 standard supports implicit casting between similar data types:

- Number types
- CHAR, VARCHAR, LONG VARCHAR
- BINARY, VARBINARY, LONG VARBINARY

Vertica supports two types of nonstandard implicit casts:

- From CHAR to FLOAT, to match the one from VARCHAR to FLOAT. The following example converts the CHAR '3' to a FLOAT so it can add the number 4.33 to the FLOAT result of the second expression:

```
=> SELECT '3'::CHAR + 4.33::FLOAT;
?column?
-----
      7.33
(1 row)
```

- Between DATE and TIMESTAMP. The following example DATE to a TIMESTAMP and calculates the time 6 hours, 6 minutes, and 6 seconds back from 12:00 AM:

```
=> SELECT DATE('now') - INTERVAL '6:6:6';
?column?
-----
2013-07-30 17:53:54
(1 row)
```

When there is no ambiguity about the data type of an expression value, it is implicitly coerced to match the expected data type. In the following command, the quoted string constant '2' is implicitly coerced into an INTEGER value so that it can be the operand of an arithmetic operator (addition):

```
=> SELECT 2 + '2';
?column?
-----
      4
(1 row)
```

A concatenate operation explicitly takes arguments of any data type. In the following example, the concatenate operation implicitly coerces the arithmetic expression `2 + 2` and the INTEGER constant `2` to VARCHAR values so that they can be concatenated.

```
=> SELECT 2 + 2 || 2;
?column?
-----
      42
(1 row)
```

Another example is to first get today's date:

```
=> SELECT DATE 'now';
?column?
-----
2013-07-31
(1 row)
```

The following command converts DATE to a TIMESTAMP and adds a day and a half to the results by using INTERVAL:

```
=> SELECT DATE 'now' + INTERVAL '1 12:00:00';
       ?column?
-----
2013-07-31 12:00:00
(1 row)
```

Most implicit casts stay within their relational family and go in one direction, from less detailed to more detailed. For example:

- DATE to TIMESTAMP/TZ
- INTEGER to NUMERIC to FLOAT
- CHAR to FLOAT
- CHAR to VARCHAR
- CHAR and/or VARCHAR to FLOAT
- CHAR to LONG VARCHAR
- VARCHAR to LONG VARCHAR
- BINARY to VARBINARY
- BINARY to LONG VARBINARY
- VARBINARY to LONG VARBINARY

More specifically, data type coercion works in this manner in Vertica:

Type	Direction	Type	Notes
INT8	>	FLOAT8	Implicit, can lose significance
FLOAT8	>	INT8	Explicit, rounds
VARCHAR	<->	CHAR	Implicit, adjusts trailing spaces
VARBINARY	<->	BINARY	Implicit, adjusts trailing NULs
VARCHAR	>	LONG VARCHAR	Implicit, adjusts trailing spaces
VARBINARY	>	LONG VARBINARY	Implicit, adjusts trailing NULs

No other types cast to or from LONGVARBINARY, VARBINARY, or BINARY. In the following list, <any> means one these types: INT8, FLOAT8, DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL.

- <any> -> VARCHAR—implicit
- VARCHAR -> <any>—explicit, except that VARCHAR->FLOAT is implicit
- <any> <-> CHAR—explicit
- DATE -> TIMESTAMP/TZ—implicit
- TIMESTAMP/TZ -> DATE—explicit, loses time-of-day
- TIME -> TIMETZ—implicit, adds local timezone
- TIMETZ -> TIME—explicit, loses timezone
- TIME -> INTERVAL—implicit, day to second with days=0
- INTERVAL -> TIME—explicit, truncates non-time parts
- TIMESTAMP <-> TIMESTAMPTZ—implicit, adjusts to local timezone
- TIMESTAMP/TZ -> TIME—explicit, truncates non-time parts
- TIMESTAMPTZ -> TIMETZ—explicit
- VARBINARY -> LONG VARBINARY—implicit
- LONG VARBINARY -> VARBINARY—explicit
- VARCHAR -> LONG VARCHAR—implicit
- LONG VARCHAR -> VARCHAR—explicit

Important: Implicit casts from INTEGER, FLOAT, and NUMERIC to VARCHAR are not supported. If you need that functionality, write an explicit cast:

```
CAST(x AS data-type-name)
```

or

```
x::data-type-name
```

The following example casts a FLOAT to an INTEGER:

```
=> SELECT(FLOAT '123.5')::INT;  
?column?
```

```
-----  
      124  
(1 row)
```

String-to-numeric data type conversions accept formats of quoted constants for scientific notation, binary scaling, hexadecimal, and combinations of numeric-type literals:

- Scientific notation :

```
=> SELECT FLOAT '1e10';  
      ?column?  
-----  
10000000000  
(1 row)
```

- BINARY scaling :

```
=> SELECT NUMERIC '1p10';  
      ?column?  
-----  
      1024  
(1 row)
```

- Hexadecimal:

```
=> SELECT NUMERIC '0x0abc';  
      ?column?  
-----  
      2748  
(1 row)
```

Examples

The following example casts three strings as NUMERICs:

```
=> SELECT NUMERIC '12.3e3', '12.3p10'::NUMERIC, CAST('0x12.3p-10e3' AS NUMERIC);  
?column? | ?column? |      ?column?  
-----+-----+-----  
12300 | 12595.2 | 17.76123046875000  
(1 row)
```

This example casts a VARBINARY string into a LONG VARBINARY data type:

```
=> SELECT B'101111000'::LONG VARBINARY;  
?column?  
-----  
\001x
```


Conversion Types					
Data Types	Implicit	Explicit	Assignment	Assignment without numeric meaning	Conversion without explicit casting
BOOLEAN			INTEGER LONG VARCHAR VARCHAR CHAR		
INTEGER	BOOLEAN NUMERIC FLOAT		INTERVAL DAY/SECOND INTERVAL YEAR/MONTH	LONG VARCHAR VARCHAR CHAR	
NUMERIC	FLOAT		INTEGER	LONG VARCHAR VARCHAR CHAR	NUMERIC
FLOAT			INTEGER NUMERIC	LONG VARCHAR VARCHAR CHAR	
LONG VARCHAR	FLOAT CHAR	BOOLEAN INTEGER NUMERIC VARCHAR TIMESTAMP TIMESTAMP WITH TIME ZONE DATE TIME TIME WITH TIME ZONE INTERVAL DAY/SECOND INTERVAL YEAR/MONTH LONG VARBINARY			LONG VARCHAR
VARCHAR	FLOAT LONG VARCHAR CHAR	BOOLEAN INTEGER NUMERIC TIMESTAMP TIMESTAMP WITH TIME ZONE DATE TIME TIME WITH TIME ZONE INTERVAL			VARCHAR

Conversion Types					
Data Types	Implicit	Explicit	Assignment	Assignment without numeric meaning	Conversion without explicit casting
		DAY/SECOND INTERVAL YEAR/MONTH			
CHAR	FLOAT LONG VARCHAR VARCHAR	BOOLEAN INTEGER NUMERIC TIMESTAMP TIMESTAMP WITH TIME ZONE DATE TIME TIME WITH TIME ZONE INTERVAL DAY/SECOND INTERVAL YEAR/MONTH			CHAR
TIMESTAMP	TIMESTAMP WITH TIME ZONE		LONG CHAR VARCHAR CHAR DATE TIME		TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP		LONG CHAR VARCHAR CHAR DATE TIME TIME WITH TIME ZONE		TIMESTAMP WITH TIME ZONE
DATE	TIMESTAMP		LONG CHAR VARCHAR CHAR TIMESTAMP WITH TIME ZONE		
TIME	TIME WITH TIME ZONE	TIMESTAMP TIMESTAMP WITH TIME ZONE INTERVAL DAY/SECOND	LONG CHAR VARCHAR CHAR		TIME
TIME WITH TIME ZONE		TIMESTAMP TIMESTAMP WITH TIME ZONE	LONG CHAR VARCHAR CHAR		TIME WITH TIME ZONE

Conversion Types					
Data Types	Implicit	Explicit	Assignment	Assignment without numeric meaning	Conversion without explicit casting
			TIME		
INTERVAL DAY/SECOND		TIME	INTEGER LONG CHAR VARCHAR CHAR		INTERVAL DAY/SECOND
INTERVAL YEAR/MONTH			INTEGER LONG CHAR VARCHAR CHAR		INTERVAL YEAR/MONTH
LONG VARBINARY		VARBINARY			LONG VARBINARY
VARBINARY	LONG VARBINARY BINARY				VARBINARY
BINARY	VARBINARY				BINARY

Implicit and Explicit Conversion

Vertica supports data type conversion of values without explicit casting, such as `NUMERIC(10,6) -> NUMERIC(18,4)`. Implicit data type conversion occurs automatically when converting values of different, but compatible, types to the target column's data type. For example, when adding values, `(INT + NUMERIC)`, the result is implicitly cast to a `NUMERIC` type to accommodate the prominent type in the statement. Depending on the input data types, different precision and scale can occur.

An explicit type conversion must occur when the source data cannot be cast implicitly to the target column's data type.

Assignment Conversion

In data assignment conversion, coercion implicitly occurs when values are assigned to database columns in an `INSERT` or `UPDATE . . SET` statement. For example, in a statement that includes

INSERT ... VALUES('2.5'), where the target column data type is NUMERIC(18,5), a cast from VARCHAR to the column data type is inferred.

In an assignment without numeric meaning, the value is subject to CHAR/VARCHAR/LONG VARCHAR comparisons.

See Also

- [Data Type Coercion](#)
- [Data Type Coercion Operators \(CAST\)](#)

SQL Functions

Functions return information from the database. Except for [Vertica-specific functions](#), you can use a function anywhere an expression is allowed.

This chapter describes each function that Vertica supports. The Behavior Type section of these descriptions categorizes the function's return behavior as one of the following:

Immutable (invariant)

When run with a given set of arguments, immutable functions such as [AVG\(\)](#) always produce the same result, regardless of environment or session settings such as locale. Some immutable functions can take an optional stable argument; in this case they are treated as stable functions.

Stable

When run with a given set of arguments, stable functions produce the same result within a single query or scan operation. However, a stable function can produce different results when issued under different environments or at different times, such as change of locale and time zone—for example, [SYSDATE\(\)](#) and 'today'.

Volatile

Regardless of their arguments or environment, volatile functions can return a different result with each invocation—for example, [RANDOM\(\)](#).

Note: All Vertica-specific functions are volatile; thus, the descriptions of these functions omit a Behavior Type section.

Aggregate Functions

Note: All functions in this section that have an [analytic](#) function counterpart are appended with [Aggregate] to avoid confusion between the two.

Aggregate functions summarize data over groups of rows from a query result set. The groups are specified using the [GROUP BY](#) clause. They are allowed only in the select list and in the [HAVING](#) and [ORDER BY](#) clauses of a [SELECT](#) statement (as described in [Aggregate Expressions](#)).

Notes

- Except for `COUNT`, these functions return a null value when no rows are selected. In particular, `SUM` of no rows returns `NULL`, not zero.
- In some cases you can replace an expression that includes multiple aggregates with a single aggregate of an expression. For example `SUM(x) + SUM(y)` can be expressed as `SUM(x+y)` (where `x` and `y` are `NOT NULL`).
- Vertica does not support nested aggregate functions.

You can also use some of the simple aggregate functions as analytic (window) functions. See [Analytic Functions](#) for details. See also [SQL Analytics](#) in Analyzing Data.

APPROXIMATE_COUNT_DISTINCT

Returns the number of distinct non-NULL values in a data set.

Behavior Type

Immutable

Syntax

```
APPROXIMATE_COUNT_DISTINCT ( expression [, error-tolerance ] )
```

Parameters

<i>expression</i>	Value to be evaluated using any data type that supports equality comparison.
<i>error-tolerance</i>	<p>Numeric value that represents the desired percentage of error tolerance, distributed around the value returned by this function. The smaller the error tolerance, the closer the approximation</p> <p>You can set <i>error-tolerance</i> to a minimum value of 0.88, with an error than is lognormally distributed with standard deviation. Vertica imposes no maximum restriction, but any value greater than 5 is implemented with 5% error tolerance.</p> <p>If you omit this argument, Vertica uses an error tolerance of 1.75 (%).</p>

Restrictions

APPROXIMATE_COUNT_DISTINCT and DISTINCT aggregates cannot be in the same query block.

Error Tolerance

APPROXIMATE_COUNT_DISTINCT(*x*, *error-tolerance*) returns a value equal to COUNT (DISTINCT *x*), with an error that is lognormally distributed with standard deviation.

Parameter *error-tolerance* is optional. Supply this argument to specify the desired standard deviation. *error-tolerance* is defined as 2.17 standard deviations, which corresponds to a 97 percent confidence interval:

$$\text{standard-deviation} = \text{error tolerance} / 2.17$$

For example:

- ***error-tolerance* = 1**
The default setting, corresponds to a standard deviation. 97 percent of the time, APPROXIMATE_COUNT_DISTINCT(*x*, 5) returns a value between:

- `COUNT(DISTINCT x) / 1.01`
- `COUNT(DISTINCT x) * 1.01`
- ***error-tolerance* = 5**
97 percent of the time, `APPROXIMATE_COUNT_DISTINCT(x)` returns a value between:
 - `COUNT(DISTINCT x) / 1.05`
 - `COUNT(DISTINCT x) * 1.05`

A 99 percent confidence interval corresponds to 2.58 standard deviations. To set *error-tolerance* confidence level corresponding to 99 (instead of a 97) percent, multiply *error-tolerance* by $2.17 / 2.58 = 0.841$.

For example, if you specify *error-tolerance* as $5 * 0.841 = 4.2$, `APPROXIMATE_COUNT_DISTINCT(x, 4.2)` returns values 99 percent of the time between:

- `COUNT (DISTINCT x)COUNT / 1.05`
- `COUNT (DISTINCT x) * 1.05`

Examples

Count the total number of distinct values in column `product_key` from table `store.store_sales_fact`:

```
=> SELECT COUNT(DISTINCT product_key) FROM store.store_sales_fact;
COUNT
-----
19982
(1 row)
```

Count the approximate number of distinct values in `product_key` with various error tolerances. The smaller the error tolerance, the closer the approximation:

```
=> SELECT APPROXIMATE_COUNT_DISTINCT(product_key,5) AS five_pct_accuracy,
APPROXIMATE_COUNT_DISTINCT(product_key,1) AS one_pct_accuracy,
APPROXIMATE_COUNT_DISTINCT(product_key,.88) AS point_eighteight_pct_accuracy
FROM store.store_sales_fact;

five_pct_accuracy | one_pct_accuracy | point_eighteight_pct_accuracy
-----+-----+-----
19431 | 19921 | 19921
(1 row)
```

See Also

- [APPROXIMATE_COUNT_DISTINCT_SYNOPSIS](#)
- [APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS](#)
- [COUNT \[Aggregate\]](#)

APPROXIMATE_COUNT_DISTINCT_SYNOPSIS

Summarizes the information of distinct non-NULL values and materializes the result set in a `VARBINARY` or `LONG VARBINARY` *synopsis* object. The calculated result is within a specified range of error tolerance. You save the synopsis object in a Vertica table for use by [APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS](#).

Note: If you are upgrading from a release to Vertica 8.1.1, you must drop the pre-existing synopsis object and recreate it to get the result.

Behavior Type

Immutable

Syntax

```
APPROXIMATE_COUNT_DISTINCT_SYNOPSIS ( expression [, error-tolerance] )
```

Parameters

<i>expression</i>	Value to evaluate using any data type that supports equality comparison.
<i>error-tolerance</i>	Numeric value that represents the desired percentage of error tolerance, distributed around the value returned by this function. The smaller the error tolerance, the closer the approximation You can set <i>error-tolerance</i> to a minimum value of 0.88, with an error than is lognormally distributed with standard deviation.

	<p>Vertica imposes no maximum restriction, but any value greater than 5 is implemented with 5% error tolerance.</p> <p>If you omit this argument, Vertica uses an error tolerance of 1.75 (%).</p> <p>For more details, see APPROXIMATE_COUNT_DISTINCT.</p>
--	---

Restrictions

`APPROXIMATE_COUNT_DISTINCT_SYNOPSIS` and `DISTINCT` aggregates cannot be in the same query block.

Example

See [APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS](#).

APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS

Calculates the number of distinct non-NULL values from the synopsis objects created by [APPROXIMATE_COUNT_DISTINCT_SYNOPSIS](#).

Note: If you are upgrading from a release to Vertica 8.1.1, you must drop the pre-existing synopsis object and recreate it to get the result.

Behavior Type

Immutable

Syntax

```
APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS ( synopsis-obj[, error-tolerance ] )
```

Parameters

<i>synopsis-obj</i>	A synopsis object created by APPROXIMATE_COUNT_DISTINCT_SYNOPSIS .
---------------------	--

<i>error-tolerance</i>	<p>Numeric value that represents the desired percentage of error tolerance, distributed around the value returned by this function. The smaller the error tolerance, the closer the approximation</p> <p>You can set <i>error-tolerance</i> to a minimum value of 0.88, with an error than is lognormally distributed with standard deviation. Vertica imposes no maximum restriction, but any value greater than 5 is implemented with 5% error tolerance.</p> <p>If you omit this argument, Vertica uses an error tolerance of 1.75 (%).</p> <p>For more details, see APPROXIMATE_COUNT_DISTINCT.</p>
------------------------	---

Restrictions

`APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS` and `DISTINCT` aggregates cannot be in the same query block.

Examples

The following examples review and compare different ways to obtain a count of unique values in a table column:

Return an exact count of unique values in column `product_key`, from table `store.store_sales_fact`:

```
=> \timing
Timing is on.
=> SELECT COUNT(DISTINCT product_key) from store.store_sales_fact;
count
-----
19982
(1 row)

Time: First fetch (1 row): 553.033 ms. All rows formatted: 553.075 ms
```

Return an approximate count of unique values in column `product_key`:

```
=> SELECT APPROXIMATE_COUNT_DISTINCT(product_key) as unique_product_keys
FROM store.store_sales_fact;
unique_product_keys
-----
19921
(1 row)

Time: First fetch (1 row): 394.562 ms. All rows formatted: 394.600 ms
```

Create a synopsis object that represents a set of `store.store_sales_fact` data with unique `product_key` values, store the synopsis in the new table `my_summary`:

```
=> CREATE TABLE my_summary AS SELECT APPROXIMATE_COUNT_DISTINCT_SYNOPSIS (product_key) syn
    FROM store.store_sales_fact;
CREATE TABLE
Time: First fetch (0 rows): 582.662 ms. All rows formatted: 582.682 ms
```

Return a count from the saved synopsis:

```
=> SELECT APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS(syn) FROM my_summary;
ApproxCountDistinctOfSynopsis
-----
                        19921
(1 row)

Time: First fetch (1 row): 105.295 ms. All rows formatted: 105.335 ms
```

APPROXIMATE_MEDIAN [Aggregate]

Computes the approximate median of an expression over a group of rows. The function returns a FLOAT value.

APPROXIMATE_MEDIAN is an alias of [APPROXIMATE_PERCENTILE \[Aggregate\]](#) with a parameter of 0.5.

Note: This function is best suited for large groups of data. If you have a small group of data, use the exact [MEDIAN \[Analytic\]](#) function.

Behavior Type

Immutable

Syntax

```
APPROXIMATE_MEDIAN ( expression )
```

Parameters

<i>expression</i>	Any FLOAT or INT data type. The function returns the approximate middle value or an interpolated value that would be the approximate
-------------------	--

middle value once the values are sorted. Null values are ignored in the calculation.

Examples

Tip: For optimal performance when using `GROUP BY` in your query, verify that your table is sorted on the `GROUP BY` column.

The following examples uses this table:

```
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT) ORDER BY state;
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

Calculate the approximate median of all sales in this table:

```
=> SELECT APPROXIMATE_MEDIAN (sales) FROM allsales;
APROXIMATE_MEDIAN
-----
                20
(1 row)
```

Modify the query to group sales by state, and obtain the approximate median for each one:

```
=> SELECT state, APPROXIMATE_MEDIAN(sales) FROM allsales GROUP BY state;
state | APPROXIMATE_MEDIAN
-----+-----
MA    |                    35
NY    |                    20
(2 rows)
```

See Also

- [MEDIAN \[Analytic\]](#)
- [PERCENTILE_CONT \[Analytic\]](#)

- [APPROXIMATE_PERCENTILE \[Aggregate\]](#)
- [SQL Analytics](#)

APPROXIMATE_PERCENTILE [Aggregate]

Computes the approximate percentile of an expression over a group of rows. This function returns a FLOAT value. Nulls are ignored.

Note: This function is best suited for large groups of data. If you have a small group of data, use the exact [PERCENTILE_CONT \[Analytic\]](#) function.

Behavior Type

Immutable

Syntax

```
APPROXIMATE_PERCENTILE ( expression USING PARAMETERS percentile = number )
```

Parameters

<i>expression</i>	Any FLOAT or INT data type. Null values are ignored.
<i>number</i>	Percentile value, which must be a FLOAT constant ranging from 0 to 1 (inclusive).

Examples

Tip: For optimal performance when using GROUP BY in your query, verify that your table is sorted on the GROUP BY column.

The following example uses this table:

```
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT) ORDER BY state;
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

Calculate the approximate percentile for sales in each state:

```
=> SELECT state, APPROXIMATE_PERCENTILE(sales USING PARAMETERS percentile=0.5) AS median FROM
allsales GROUP BY state;
state | median
-----+-----
MA    |      35
NY    |      20
(2 rows)
```

See Also

- [MEDIAN \[Analytic\]](#)
- [PERCENTILE_CONT \[Analytic\]](#)
- [SQL Analytics](#)

AVG [Aggregate]

Computes the average (arithmetic mean) of an expression over a group of rows. AVG always returns a DOUBLE PRECISION value.

The AVG aggregate function differs from the [AVG](#) analytic function, which computes the average of an expression over a group of rows within a window.

Behavior Type

Immutable

Syntax

```
AVG ( [ ALL | DISTINCT ] expression )
```

Parameters

ALL	Invokes the aggregate function for all rows in the group (default).
DISTINCT	Invokes the aggregate function for all distinct non-null values of the expression found in the group.
<i>expression</i>	The value whose average is calculated over a set of rows, any expression that can have a DOUBLE PRECISION result.

Overflow Handling

By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

Examples

The following query returns the average income from the customer table:

```
=> SELECT AVG(annual_income) FROM customer_dimension;
      AVG
-----
2104270.6485
(1 row)
```

See Also

- [COUNT \[Aggregate\]](#)
- [SUM \[Aggregate\]](#)
- [Numeric Data Types](#)

BIT_AND

Takes the bitwise AND of all non-null input values. If the input parameter is NULL, the return value is also NULL.

Behavior Type

Immutable

Syntax

```
BIT_AND ( expression )
```

Parameters

<i>expression</i>	The BINARY or VARBINARY input value to evaluate. BIT_AND operates on VARBINARY types explicitly and on BINARY types implicitly through casts .
-------------------	--

Returns

BIT_AND returns:

- The same value as the argument data type.
- 1 for each bit compared, if all bits are 1; otherwise 0.

If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing hex values ff, null, and f, BIT_AND ignores the null value and extends the value f to f0.

Example

The example that follows uses table t with a single column of VARBINARY data type:

```
=> CREATE TABLE t ( c VARBINARY(2) );  
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
```

```
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));  
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table `t` to see column `c` output:

```
=> SELECT TO_HEX(c) FROM t;  
TO_HEX  
-----  
ff00  
ffff  
f00f  
(3 rows)
```

Query table `t` to get the AND value for column `c`:

```
=> SELECT TO_HEX(BIT_AND(c)) FROM t;  
TO_HEX  
-----  
f000  
(1 row)
```

The function is applied pairwise to all values in the group, resulting in `f000`, which is determined as follows:

1. `ff00` (record 1) is compared with `ffff` (record 2), which results in `ff00`.
2. The result from the previous comparison is compared with `f00f` (record 3), which results in `f000`.

See Also

[Binary Data Types](#)

BIT_OR

Takes the bitwise OR of all non-null input values. If the input parameter is NULL, the return value is also NULL.

Behavior Type

Immutable

Syntax

`BIT_OR (expression)`

Parameters

<i>expression</i>	The [BINARY VARBINARY] input value to be evaluated. BIT_OR() operates on VARBINARY types explicitly and on BINARY types implicitly through casts .
-------------------	--

Returns

BIT_OR returns:

- The same value as the argument data type.
- 1 for each bit compared, if any bit is 1; otherwise 0.

If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing hex values ff, null, and f, the function ignores the null value and extends the value f to f0.

Example

The example that follows uses table t with a single column of VARBINARY data type:

```
=> CREATE TABLE t ( c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table t to see column c output:

```
=> SELECT TO_HEX(c) FROM t;
TO_HEX
-----
ff00
ffff
f00f
(3 rows)
```

Query table t to get the OR value for column c:

```
=> SELECT TO_HEX(BIT_OR(c)) FROM t;  
TO_HEX  
-----  
ffff  
(1 row)
```

The function is applied pairwise to all values in the group, resulting in ffff, which is determined as follows:

1. ff00 (record 1) is compared with ffff, which results in ffff.
2. The ff00 result from the previous comparison is compared with f00f (record 3), which results in ffff.

See Also

[Binary Data Types](#)

BIT_XOR

Takes the bitwise XOR of all non-null input values. If the input parameter is NULL, the return value is also NULL.

Behavior Type

Immutable

Syntax

```
BIT_XOR ( expression )
```

Parameters

<i>expression</i>	The BINARY or VARBINARY input value to evaluate. BIT_XOR operates on VARBINARY types explicitly and on BINARY types implicitly through casts .
-------------------	--

Returns

BIT_XOR returns:

- The same value as the argument data type.
- 1 for each bit compared, if there are an odd number of arguments with set bits; otherwise 0.

If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing hex values ff, null, and f, the function ignores the null value and extends the value f to f0.

Example

First create a sample table and projections with binary columns:

The example that follows uses table t with a single column of VARBINARY data type:

```
=> CREATE TABLE t ( c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table t to see column c output:

```
=> SELECT TO_HEX(c) FROM t;
 TO_HEX
-----
ff00
ffff
f00f
(3 rows)
```

Query table t to get the XOR value for column c:

```
=> SELECT TO_HEX(BIT_XOR(c)) FROM t;
 TO_HEX
-----
f0f0
(1 row)
```

See Also

[Binary Data Types](#)

BOOL_AND [Aggregate]

Processes Boolean values and returns a Boolean value result. If all input values are true, BOOL_AND returns t. Otherwise it returns f (false).

Behavior Type

Immutable

Syntax

```
BOOL_AND ( expression )
```

Parameters

<i>expression</i>	A Boolean data type or any non-Boolean data type that can be implicitly coerced to a Boolean data type.
-------------------	---

Examples

The following example shows how to use aggregate functions BOOL_AND, BOOL_OR, and BOOL_XOR. The sample table mixers includes columns for models and colors.

```
=> CREATE TABLE mixers(model VARCHAR(20), colors VARCHAR(20));  
CREATE TABLE
```

Insert sample data into the table. The sample adds two color fields for each model.

```
=> INSERT INTO mixers  
SELECT 'beginner', 'green'  
UNION ALL  
SELECT 'intermediate', 'blue'  
UNION ALL  
SELECT 'intermediate', 'blue'  
UNION ALL  
SELECT 'advanced', 'green'  
UNION ALL  
SELECT 'advanced', 'blue'  
UNION ALL  
SELECT 'professional', 'blue'
```

```
UNION ALL
SELECT 'professional', 'green'
UNION ALL
SELECT 'beginner', 'green';
OUTPUT
-----
      8
(1 row)
```

Query the table. The result shows models that have two blue (BOOL_AND), one or two blue (BOOL_OR), and specifically not more than one blue (BOOL_XOR) mixer.

```
=> SELECT model,
BOOL_AND(colors= 'blue')AS two_blue,
BOOL_OR(colors= 'blue')AS one_or_two_blue,
BOOL_XOR(colors= 'blue')AS specifically_not_more_than_one_blue
FROM mixers
GROUP BY model;
```

model	two_blue	one_or_two_blue	specifically_not_more_than_one_blue
advanced	f	t	t
beginner	f	f	f
intermediate	t	t	f
professional	f	t	t

(4 rows)

See Also

- [BOOL_AND \[Analytic\]](#)
- [BOOL_OR \[Aggregate\]](#)
- [BOOL_XOR \[Aggregate\]](#)
- [Boolean Data Type](#)

BOOL_OR [Aggregate]

Processes Boolean values and returns a Boolean value result. If at least one input value is true, BOOL_OR returns t. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

`BOOL_OR (expression)`

Parameters

<i>expression</i>	A Boolean data type or any non-Boolean data type that can be implicitly coerced to a Boolean data type.
-------------------	---

Examples

The following example shows how to use aggregate functions `BOOL_AND`, `BOOL_OR`, and `BOOL_XOR`. The sample table `mixers` includes columns for models and colors.

```
=> CREATE TABLE mixers(model VARCHAR(20), colors VARCHAR(20));  
CREATE TABLE
```

Insert sample data into the table. The sample adds two color fields for each model.

```
=> INSERT INTO mixers  
SELECT 'beginner', 'green'  
UNION ALL  
SELECT 'intermediate', 'blue'  
UNION ALL  
SELECT 'intermediate', 'blue'  
UNION ALL  
SELECT 'advanced', 'green'  
UNION ALL  
SELECT 'advanced', 'blue'  
UNION ALL  
SELECT 'professional', 'blue'  
UNION ALL  
SELECT 'professional', 'green'  
UNION ALL  
SELECT 'beginner', 'green';  
OUTPUT  
-----  
      8  
(1 row)
```

Query the table. The result shows models that have two blue (`BOOL_AND`), one or two blue (`BOOL_OR`), and specifically not more than one blue (`BOOL_XOR`) mixer.

```
=> SELECT model,  
       BOOL_AND(colors= 'blue')AS two_blue,  
       BOOL_OR(colors= 'blue')AS one_or_two_blue,
```

```
BOOL_XOR(colors= 'blue')AS specifically_not_more_than_one_blue  
FROM mixers  
GROUP BY model;
```

model	two_blue	one_or_two_blue	specifically_not_more_than_one_blue
advanced	f	t	t
beginner	f	f	f
intermediate	t	t	f
professional	f	t	t

(4 rows)

See Also

- [BOOL_OR \[Analytic\]](#)
- [BOOL_AND \[Aggregate\]](#)
- [BOOL_XOR \[Aggregate\]](#)
- [Boolean Data Type](#)

BOOL_XOR [Aggregate]

Processes Boolean values and returns a Boolean value result. If specifically only one input value is true, BOOL_XOR returns t. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

```
BOOL_XOR ( expression )
```

Parameters

<i>expression</i>	A Boolean data type or any non-Boolean data type that can be implicitly coerced to a Boolean data type.
-------------------	---

Examples

The following example shows how to use aggregate functions `BOOL_AND`, `BOOL_OR`, and `BOOL_XOR`. The sample table `mixers` includes columns for models and colors.

```
=> CREATE TABLE mixers(model VARCHAR(20), colors VARCHAR(20));  
CREATE TABLE
```

Insert sample data into the table. The sample adds two color fields for each model.

```
=> INSERT INTO mixers  
SELECT 'beginner', 'green'  
UNION ALL  
SELECT 'intermediate', 'blue'  
UNION ALL  
SELECT 'intermediate', 'blue'  
UNION ALL  
SELECT 'advanced', 'green'  
UNION ALL  
SELECT 'advanced', 'blue'  
UNION ALL  
SELECT 'professional', 'blue'  
UNION ALL  
SELECT 'professional', 'green'  
UNION ALL  
SELECT 'beginner', 'green';  
OUTPUT  
-----  
      8  
(1 row)
```

Query the table. The result shows models that have two blue (`BOOL_AND`), one or two blue (`BOOL_OR`), and specifically not more than one blue (`BOOL_XOR`) mixer.

```
=> SELECT model,  
BOOL_AND(colors= 'blue')AS two_blue,  
BOOL_OR(colors= 'blue')AS one_or_two_blue,  
BOOL_XOR(colors= 'blue')AS specifically_not_more_than_one_blue  
FROM mixers  
GROUP BY model;  
  
   model   | two_blue | one_or_two_blue | specifically_not_more_than_one_blue  
-----+-----+-----+-----  
advanced   | f        | t                | t  
beginner   | f        | f                | f  
intermediate | t        | t                | f  
professional | f        | t                | t  
(4 rows)
```

See Also

- [BOOL_XOR \[Analytic\]](#)
- [BOOL_AND \[Aggregate\]](#)
- [BOOL_OR \[Aggregate\]](#)
- [Boolean Data Type](#)

CORR

Returns the DOUBLE PRECISION coefficient of correlation of a set of expression pairs. CORR eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

Syntax

```
CORR ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT CORR (Annual_salary, Employee_age) FROM employee_dimension;  
CORR  
-----  
-0.00719153413192422  
(1 row)
```

COUNT [Aggregate]

Returns as a BIGINT the number of rows in each group where the expression is not NULL. If the query has no GROUP BY clause, COUNT returns the number of table rows.

The COUNT aggregate function differs from the [COUNT](#) analytic function, which returns the number over a group of rows within a window.

Behavior Type

Immutable

Syntax

```
COUNT ( [ * ] [ ALL | DISTINCT ] expression )
```

Parameters

<code>*</code>	Specifies to count all rows in the specified table or each group.
<code>ALL DISTINCT</code>	Specifies how to count rows where <i>expression</i> has a non-null value: <ul style="list-style-type: none">• ALL (default): Counts all rows where <i>expression</i> evaluates to a non-null value.• DISTINCT: Counts all rows where <i>expression</i> evaluates to a distinct non-null value.
<i>expression</i>	The column or expression whose non-null values are counted.

Examples

The following query returns the number of distinct values in the `primary_key` column of the `date_dimension` table:

```
=> SELECT COUNT (DISTINCT date_key) FROM date_dimension;
```

```
COUNT
-----
 1826
(1 row)
```

This example returns all distinct values of evaluating the expression `x+y` for all `inventory_fact` records.

```
=> SELECT COUNT (DISTINCT date_key + product_key) FROM inventory_fact;

COUNT
-----
21560
(1 row)
```

You can create an equivalent query using the `LIMIT` keyword to restrict the number of rows returned:

```
=> SELECT COUNT(date_key + product_key) FROM inventory_fact GROUP BY date_key LIMIT 10;

COUNT
-----
 173
   31
  321
  113
  286
   84
  244
  238
  145
  202
(10 rows)
```

This query returns the number of distinct values of `date_key` in all records with the specific distinct `product_key` value.

```
=> SELECT product_key, COUNT (DISTINCT date_key) FROM inventory_fact
   GROUP BY product_key LIMIT 10;

product_key | count
-----+-----
          1 |    12
          2 |    18
          3 |    13
          4 |    17
          5 |    11
          6 |    14
          7 |    13
          8 |    17
          9 |    15
         10 |    12
(10 rows)
```

This query counts each distinct `product_key` value in `inventory_fact` table with the constant `1`.

```
=> SELECT product_key, COUNT (DISTINCT product_key) FROM inventory_fact
      GROUP BY product_key LIMIT 10;
```

product_key	count
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

(10 rows)

This query selects each distinct date_key value and counts the number of distinct product_key values for all records with the specific product_key value. It then sums the qty_in_stock values in all records with the specific product_key value and groups the results by date_key.

```
=> SELECT date_key, COUNT (DISTINCT product_key), SUM(qty_in_stock) FROM inventory_fact
      GROUP BY date_key LIMIT 10;
```

date_key	count	sum
1	173	88953
2	31	16315
3	318	156003
4	113	53341
5	285	148380
6	84	42421
7	241	119315
8	238	122380
9	142	70151
10	202	95274

(10 rows)

This query selects each distinct product_key value and then counts the number of distinct date_key values for all records with the specific product_key value. It also counts the number of distinct warehouse_key values in all records with the specific product_key value.

```
=> SELECT product_key, COUNT (DISTINCT date_key), COUNT (DISTINCT warehouse_key) FROM inventory_fact
      GROUP BY product_key LIMIT 15;
```

product_key	count	count
1	12	12
2	18	18
3	13	12
4	17	18
5	11	9
6	14	13
7	13	13

```

      8 |    17 |    15
      9 |    15 |    14
     10 |    12 |    12
     11 |    11 |    11
     12 |    13 |    12
     13 |     9 |     7
     14 |    13 |    13
     15 |    18 |    17
(15 rows)

```

This query selects each distinct `product_key` value, counts the number of distinct `date_key` and `warehouse_key` values for all records with the specific `product_key` value, and then sums all `qty_in_stock` values in records with the specific `product_key` value. It then returns the number of `product_version` values in records with the specific `product_key` value.

```

=> SELECT product_key, COUNT (DISTINCT date_key),
      COUNT (DISTINCT warehouse_key),
      SUM (qty_in_stock),
      COUNT (product_version)
      FROM inventory_fact GROUP BY product_key LIMIT 15;

```

```

product_key | count | count | sum | count
-----+-----+-----+-----+-----
      1 |    12 |    12 | 5530 |    12
      2 |    18 |    18 | 9605 |    18
      3 |    13 |    12 | 8404 |    13
      4 |    17 |    18 | 10006 |    18
      5 |    11 |     9 | 4794 |    11
      6 |    14 |    13 | 7359 |    14
      7 |    13 |    13 | 7828 |    13
      8 |    17 |    15 | 9074 |    17
      9 |    15 |    14 | 7032 |    15
     10 |    12 |    12 | 5359 |    12
     11 |    11 |    11 | 6049 |    11
     12 |    13 |    12 | 6075 |    13
     13 |     9 |     7 | 3470 |     9
     14 |    13 |    13 | 5125 |    13
     15 |    18 |    17 | 9277 |    18
(15 rows)

```

The following example returns the number of warehouses from the `warehouse_dimension` table:

```

=> SELECT COUNT(warehouse_name) FROM warehouse_dimension;

COUNT
-----
     100
(1 row)

```

This next example returns the total number of vendors:

```

=> SELECT COUNT(*) FROM vendor_dimension;

```

```
COUNT
-----
      50
(1 row)
```

See Also

- [Analytic Functions](#)
- [AVG \[Aggregate\]](#)
- [SUM \[Aggregate\]](#)
- [SQL Analytics](#)
- [APPROXIMATE_COUNT_DISTINCT](#)
- [APPROXIMATE_COUNT_DISTINCT_SYNOPSIS](#)
- [APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS](#)

COVAR_POP

Returns the population covariance for a set of expression pairs. The return value is of type `DOUBLE PRECISION`. `COVAR_POP` eliminates expression pairs where either expression in the pair is `NULL`. If no rows remain, the function returns `NULL`.

Syntax

```
SELECT COVAR_POP ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent <code>DOUBLE PRECISION</code> expression
<i>expression2</i>	The independent <code>DOUBLE PRECISION</code> expression

Example

```
=> SELECT COVAR_POP (Annual_salary, Employee_age)
      FROM employee_dimension;
      COVAR_POP
-----
-9032.34810730019
(1 row)
```

COVAR_SAMP

Returns the sample covariance for a set of expression pairs. The return value is of type DOUBLE PRECISION. COVAR_SAMP eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

Syntax

```
SELECT COVAR_SAMP ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT COVAR_SAMP (Annual_salary, Employee_age)
      FROM employee_dimension;
      COVAR_SAMP
-----
-9033.25143244343
(1 row)
```

GROUP_ID

Uniquely identifies duplicate sets for GROUP BY queries that return duplicate grouping sets. This function returns one or more integers, starting with zero (0), as identifiers.

For the number of duplicates n for a particular grouping, GROUP_ID returns a range of sequential numbers, 0 to $n-1$. For the first each unique group it encounters, GROUP_ID returns the value 0. If GROUP_ID finds the same grouping again, the function returns 1, then returns 2 for the next found grouping, and so on.

Note: Use GROUP_ID only in SELECT statements that contain a **GROUP BY** aggregate: **CUBE**, **GROUPING SETS**, and **ROLLUP**.

Behavior Type

Immutable

Syntax

GROUP_ID ()

Examples

This example shows how GROUP_ID creates unique identifiers when a query produces duplicate groupings. For an expenses table, the following query groups the results by category of expense and year and rolls up the sum for those two columns. The results have duplicate groupings for category and NULL. The first grouping has a GROUP_ID of 0, and the second grouping has a GROUP_ID of 1.

```
=> SELECT Category, Year, SUM(Amount), GROUPING_ID(Category, Year),
      GROUP_ID() FROM expenses GROUP BY Category, ROLLUP(Category,Year)
      ORDER BY Category, Year, GROUPING_ID();
```

Category	Year	SUM	GROUPING_ID	GROUP_ID
Books	2005	39.98	0	0
Books	2007	29.99	0	0
Books	2008	29.99	0	0
Books		99.96	1	0
Books		99.96	1	1
Electricity	2005	109.99	0	0
Electricity	2006	109.99	0	0

Electricity	2007	229.98	0	0
Electricity		449.96	1	1
Electricity		449.96	1	0

See Also

- [CUBE Aggregate](#)
- [GROUPING](#)
- [GROUPING_ID](#)
- [GROUPING SETS Aggregate](#)
- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

GROUPING

Disambiguates the use of NULL values when GROUP BY queries with multilevel aggregates generate NULL values to identify subtotals in grouping columns. Such NULL values from the original data can also occur in rows. GROUPING returns 1, if the value of *expression* is:

- NULL, representing an aggregated value
- 0 for any other value, including NULL values in rows

Note: Use GROUPING only in SELECT statements that contain a [GROUP BY](#) aggregate: [CUBE](#), [GROUPING SETS](#), and [ROLLUP](#).

Behavior Type

Immutable

Syntax

```
GROUPING ( expression )
```

Parameters

<i>expression</i>	An expression in the GROUP BY clause
-------------------	--------------------------------------

Examples

The following query uses the GROUPING function, taking one of the GROUP BY expressions as an argument. For each row, GROUPING returns one of the following:

- 0: The column is part of the group for that row
- 1: The column is not part of the group for that row

The 1 in the GROUPING(Year) column for electricity and books indicates that these values are subtotals. The right-most column values for both GROUPING(Category) and GROUPING(Year) are 1. This value indicates that neither column contributed to the GROUP BY. The final row represents the total sales.

```
=> SELECT Category, Year, SUM(Amount),  
       GROUPING(Category), GROUPING(Year) FROM expenses  
       GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

Category	Year	SUM	GROUPING	GROUPING
Books	2005	39.98	0	0
Books	2007	29.99	0	0
Books	2008	29.99	0	0
Books		99.96	0	1
Electricity	2005	109.99	0	0
Electricity	2006	109.99	0	0
Electricity	2007	229.98	0	0
Electricity		449.96	0	1
		549.92	1	1

See Also

- [CUBE Aggregate](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)
- [GROUPING SETS Aggregate](#)

- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

GROUPING_ID

Concatenates the set of Boolean values generated by the [GROUPING](#) function into a bit vector. `GROUPING_ID` treats the bit vector as a binary number and returns it as a base-10 value that identifies the grouping set combination.

By using `GROUPING_ID` you avoid the need for multiple, individual `GROUPING` functions. `GROUPING_ID` simplifies row-filtering conditions, because rows of interest are identified using a single return from `GROUPING_ID = n`. Use `GROUPING_ID` to identify grouping combinations.

Note: Use `GROUPING_ID` only in `SELECT` statements that contain a [GROUP BY](#) aggregate: [CUBE](#), [GROUPING SETS](#), and [ROLLUP](#).

Behavior Type

Immutable

Syntax

```
GROUPING_ID ( [expression[,...]] )
```

<i>expression</i>	An expression that matches one of the expressions in the <code>GROUP BY</code> clause. If the <code>GROUP BY</code> clause includes a list of expressions, <code>GROUPING_ID</code> returns a number corresponding to the <code>GROUPING</code> bit vector associated with a row.
-------------------	--

Examples

This example shows how calling `GROUPING_ID` without an expression returns the `GROUPING` bit vector associated with a full set of multilevel aggregate expressions. The `GROUPING_ID`

value is comparable to `GROUPING_ID(a,b)` because `GROUPING_ID()` includes all columns in the `GROUP BY ROLLUP`:

```
=> SELECT a,b,COUNT(*), GROUPING_ID() FROM T GROUP BY ROLLUP(a,b);
```

In the following query, the `GROUPING(Category)` and `GROUPING(Year)` columns have three combinations:

- 0,0
- 0,1
- 1,1

```
=> SELECT Category, Year, SUM(Amount),
       GROUPING(Category), GROUPING(Year) FROM expenses
       GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

Category	Year	SUM	GROUPING	GROUPING
Books	2005	39.98	0	0
Books	2007	29.99	0	0
Books	2008	29.99	0	0
Books		99.96	0	1
Electricity	2005	109.99	0	0
Electricity	2006	109.99	0	0
Electricity	2007	229.98	0	0
Electricity		449.96	0	1
		549.92	1	1

`GROUPING_ID` converts these values as follows:

Binary Set Values	Decimal Equivalents
00	0
01	1
11	3
0	Category, Year

The following query returns the single number for each `GROUP BY` level that appears in the `gr_id` column:

```
=> SELECT Category, Year, SUM(Amount),
       GROUPING(Category),GROUPING(Year),GROUPING_ID(Category,Year) AS gr_id
       FROM expenses GROUP BY ROLLUP(Category, Year);
```

Category	Year	SUM	GROUPING	GROUPING	gr_id
Books	2008	29.99	0	0	0
Books	2005	39.98	0	0	0

Electricity	2007	229.98	0	0	0
Books	2007	29.99	0	0	0
Electricity	2005	109.99	0	0	0
Electricity		449.96	0	1	1
		549.92	1	1	3
Electricity	2006	109.99	0	0	0
Books		99.96	0	1	1

The `gr_id` value determines the GROUP BY level for each row:

GROUP BY Level	GROUP BY Row Level
3	Total sum
1	Category
0	Category, year

You can also use the [DECODE](#) function to give the values more meaning by comparing each search value individually:

```
=> SELECT Category, Year, SUM(AMOUNT), DECODE(GROUPING_ID(Category, Year),
      3, 'Total',
      1, 'Category',
      0, 'Category,Year')
      AS GROUP_NAME FROM expenses GROUP BY ROLLUP(Category, Year);
Category | Year | SUM | GROUP_NAME
-----+-----+-----+-----
Electricity | 2006 | 109.99 | Category,Year
Books | | 99.96 | Category
Electricity | 2007 | 229.98 | Category,Year
Books | 2007 | 29.99 | Category,Year
Electricity | 2005 | 109.99 | Category,Year
Electricity | | 449.96 | Category
| | 549.92 | Total
Books | 2005 | 39.98 | Category,Year
Books | 2008 | 29.99 | Category,Year
```

See Also

- [CUBE Aggregate](#)
- [GROUP_ID](#)
- [GROUPING](#)
- [GROUPING SETS Aggregate](#)

- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

MAX [Aggregate]

Returns the greatest value of an expression over a group of rows. The return value has the same type as the expression data type.

The MAX [analytic function](#) differs from the aggregate function, in that it returns the maximum value of an expression over a group of rows within a window.

Aggregate functions MIN and MAX can operate with Boolean values. MAX can act upon a [Boolean data type](#) or a value that can be implicitly converted to a Boolean. If at least one input value is true, MAX returns t (true). Otherwise, it returns f (false). In the same scenario, MIN returns t (true) if all input values are true. Otherwise it returns f.

Behavior Type

Immutable

Syntax

```
MAX ( expression )
```

Parameters

<i>expression</i>	Any expression for which the maximum value is calculated, typically a column reference .
-------------------	--

Examples

The following query returns the largest value in column `sales_dollar_amount`.

```
=> SELECT MAX(sales_dollar_amount) AS highest_sale FROM store.store_sales_fact;
highest_sale
-----
           600
(1 row)
```

The following example shows you the difference between the MIN and MAX aggregate functions when you use them with a Boolean value. The sample creates a table, adds two rows of data, and shows sample output for MIN and MAX.

```
=> CREATE TABLE min_max_functions (torf BOOL);

=> INSERT INTO min_max_functions VALUES (1);
=> INSERT INTO min_max_functions VALUES (0);

=> SELECT * FROM min_max_functions;
  torf
-----
  t
  f
(2 rows)

=> SELECT min(torf) FROM min_max_functions;
  min
-----
  f
(1 row)

=> SELECT max(torf) FROM min_max_functions;
  max
-----
  t
(1 row)
```

See Also

[Data Aggregation](#)

MIN [Aggregate]

Returns the smallest value of an expression over a group of rows. The return value has the same type as the expression data type.

The MIN [analytic function](#) differs from the aggregate function, in that it returns the minimum value of an expression over a group of rows within a window.

Aggregate functions MIN and MAX can operate with Boolean values. MAX can act upon a [Boolean data type](#) or a value that can be implicitly converted to a Boolean. If at least one input value is true, MAX returns t (true). Otherwise, it returns f (false). In the same scenario, MIN returns t (true) if all input values are true. Otherwise it returns f.

Behavior Type

Immutable

Syntax

`MIN (expression)`

Parameters

<i>expression</i>	Any expression for which the minimum value is calculated, typically a column reference .
-------------------	--

Examples

The following query returns the lowest salary from the `employee_dimension` table.

This example shows how you can query to return the lowest salary from the `employee_dimension` table.

```
=> SELECT MIN(annual_salary) AS lowest_paid FROM employee_dimension;
lowest_paid
-----
          1200
(1 row)
```

The following example shows you the difference between the `MIN` and `MAX` aggregate functions when you use them with a Boolean value. The sample creates a table, adds two rows of data, and shows sample output for `MIN` and `MAX`.

```
=> CREATE TABLE min_max_functions (torf BOOL);

=> INSERT INTO min_max_functions VALUES (1);
=> INSERT INTO min_max_functions VALUES (0);

=> SELECT * FROM min_max_functions;
torf
-----
t
f
(2 rows)

=> SELECT min(torf) FROM min_max_functions;
min
-----
f
(1 row)

=> SELECT max(torf) FROM min_max_functions;
max
-----
```

```
t  
(1 row)
```

See Also

[Data Aggregation](#)

REGR_AVGX

Returns the `DOUBLE PRECISION` average of the independent expression in an expression pair. `REGR_AVGX` eliminates expression pairs where either expression in the pair is `NULL`. If no rows remain, `REGR_AVGX` returns `NULL`.

Syntax

```
SELECT REGR_AVGX ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent <code>DOUBLE PRECISION</code> expression
<i>expression2</i>	The independent <code>DOUBLE PRECISION</code> expression

Example

```
=> SELECT REGR_AVGX (Annual_salary, Employee_age)  
      FROM employee_dimension;  
      REGR_AVGX  
-----  
      39.321  
(1 row)
```

REGR_AVGY

Returns the `DOUBLE PRECISION` average of the dependent expression in an expression pair. The function eliminates expression pairs where either expression in the pair is `NULL`. If no rows remain, the function returns `NULL`.

Syntax

```
REGR_AVGY ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT REGR_AVGY (Annual_salary, Employee_age)
      FROM employee_dimension;
REGR_AVGY
-----
58354.4913
(1 row)
```

REGR_COUNT

Returns the count of all rows in an expression pair. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns 0.

Syntax

```
SELECT REGR_COUNT ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT REGR_COUNT (Annual_salary, Employee_age) FROM employee_dimension;
REGR_COUNT
-----
          10000
(1 row)
```

REGR_INTERCEPT

Returns the y-intercept of the regression line determined by a set of expression pairs. The return value is of type `DOUBLE PRECISION`. `REGR_INTERCEPT` eliminates expression pairs where either expression in the pair is `NULL`. If no rows remain, `REGR_INTERCEPT` returns `NULL`.

Syntax

```
SELECT REGR_INTERCEPT ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent <code>DOUBLE PRECISION</code> expression
<i>expression2</i>	The independent <code>DOUBLE PRECISION</code> expression

Example

```
=> SELECT REGR_INTERCEPT (Annual_salary, Employee_age) FROM employee_dimension;
REGR_INTERCEPT
-----
59929.5490163437
(1 row)
```

REGR_R2

Returns the square of the correlation coefficient of a set of expression pairs. The return value is of type `DOUBLE PRECISION`. `REGR_R2` eliminates expression pairs where either expression in

the pair is NULL. If no rows remain, REGR_R2 returns NULL.

Syntax

```
SELECT REGR_R2 ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT REGR_R2 (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_R2
-----
5.17181631706311e-05
(1 row)
```

REGR_SLOPE

Returns the slope of the regression line, determined by a set of expression pairs. The return value is of type DOUBLE PRECISION. REGR_SLOPE eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_SLOPE returns NULL.

Syntax

```
SELECT REGR_SLOPE ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT REGR_SLOPE (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_SLOPE
-----
-40.056400303749
(1 row)
```

REGR_SXX

Returns the sum of squares of the difference between the independent expression (*expression2*) and its average.

That is, REGR_SXX returns: $\sum[(expression2 - average(expression2))(expression2 - average(expression2))]$

The return value is of type DOUBLE PRECISION. REGR_SXX eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_SXX returns NULL.

Syntax

```
SELECT REGR_SXX ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT REGR_SXX (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_SXX
-----
2254907.59
(1 row)
```

REGR_SXY

Returns the sum of products of the difference between the dependent expression (*expression1*) and its average and the difference between the independent expression (*expression2*) and its average.

That is, REGR_SXY returns: $\sum[(expression1 - average(expression1))(expression2 - average(expression2))]$

The return value is of type DOUBLE PRECISION. REGR_SXY eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_SXY returns NULL.

Syntax

```
SELECT REGR_SXY ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent DOUBLE PRECISION expression
<i>expression2</i>	The independent DOUBLE PRECISION expression

Example

```
=> SELECT REGR_SXY (Annual_salary, Employee_age) FROM employee_dimension;
      REGR_SXY
-----
-90323481.0730019
(1 row)
```

REGR_SYY

Returns the sum of squares of the difference between the dependent expression (*expression1*) and its average.

That is, REGR_SYY returns: $\sum[(expression1 - average(expression1))(expression1 - average(expression1))]$

The return value is of type `DOUBLE PRECISION`. `REGR_SYY` eliminates expression pairs where either expression in the pair is `NULL`. If no rows remain, `REGR_SYY` returns `NULL`.

Syntax

```
SELECT REGR_SYY ( expression1, expression2 )
```

Parameters

<i>expression1</i>	The dependent <code>DOUBLE PRECISION</code> expression
<i>expression2</i>	The independent <code>DOUBLE PRECISION</code> expression

Example

```
=> SELECT REGR_SYY (Annual_salary, Employee_age) FROM employee_dimension;  
      REGR_SYY  
-----  
69956728794707.2  
(1 row)
```

STDDEV [Aggregate]

Evaluates the statistical sample standard deviation for each member of the group. The return value is the same as the square root of [VAR_SAMP](#):

```
STDDEV(expression) = SQRT(VAR_SAMP(expression))
```

Behavior Type

Immutable

Syntax

```
STDDEV ( expression )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. STDDEV returns the same data type as <i>expression</i> .
-------------------	--

Related Functions

- Nonstandard function STDDEV is provided for compatibility with other databases. It is semantically identical to [STDDEV_SAMP](#).
- This aggregate function differs from analytic function [STDDEV](#), which computes the statistical sample standard deviation of the current row with respect to the group of rows within a window.
- When [VAR_SAMP](#) returns NULL, STDDEV returns NULL.

Examples

The following example returns the statistical sample standard deviation for each household ID from the `customer_dimension` table of the VMart example database:

```
=> SELECT STDDEV(household_id) FROM customer_dimension;  
      STDDEV  
-----  
8651.5084240071
```

STDDEV_POP [Aggregate]

Evaluates the statistical population standard deviation for each member of the group.

Behavior Type

Immutable

Syntax

`STDDEV_POP (expression)`

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. <code>STDDEV_POP</code> returns the same data type as <i>expression</i> .
-------------------	---

Related Functions

- This function differs from the analytic function [STDDEV_POP](#), which evaluates the statistical population standard deviation for each member of the group of rows within a window.
- `STDDEV_POP` returns the same value as the square root of [VAR_POP](#):

`STDDEV_POP(expression) = SQRT(VAR_POP(expression))`

- When [VAR_SAMP](#) returns NULL, this function returns NULL.

Examples

The following example returns the statistical population standard deviation for each household ID in the customer table.

```
=> SELECT STDDEV_POP(household_id) FROM customer_dimension;
   STDDEV_POP
-----
 8651.41895973367
(1 row)
```

See Also

- [Analytic Functions](#)
- [SQL Analytics](#)

STDDEV_SAMP [Aggregate]

Evaluates the statistical sample standard deviation for each member of the group. The return value is the same as the square root of [VAR_SAMP](#):

```
STDDEV_SAMP(expression) = SQRT(VAR_SAMP(expression))
```

Behavior Type

Immutable

Syntax

```
STDDEV_SAMP ( expression )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. STDDEV_SAMP returns the same data type as <i>expression</i> .
-------------------	---

Related Functions

- STDDEV_SAMP is semantically identical to nonstandard function [STDDEV](#), which is provided for compatibility with other databases.
- This aggregate function differs from analytic function [STDDEV_SAMP](#), which computes the statistical sample standard deviation of the current row with respect to the group of rows within a window.
- When [VAR_SAMP](#) returns NULL, STDDEV_SAMP returns NULL.

Examples

The following example returns the statistical sample standard deviation for each household ID from the customer dimension table.

```
=> SELECT STDDEV_SAMP(household_id) FROM customer_dimension;
      stddev_samp
-----
8651.50842400771
(1 row)
```

SUM [Aggregate]

Computes the sum of an expression over a group of rows. SUM returns a `DOUBLE PRECISION` value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

The SUM aggregate function differs from the [SUM](#) analytic function, which computes the sum of an expression over a group of rows within a window.

Behavior Type

Immutable

Syntax

```
SUM ( [ ALL | DISTINCT ] expression )
```

Parameters

ALL	Invokes the aggregate function for all rows in the group (default)
DISTINCT	Invokes the aggregate function for all distinct non-null values of the expression found in the group
<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.

Overflow Handling

If you encounter data overflow when using `SUM()`, use [SUM_FLOAT](#) which converts the data to a floating point. By default, Vertica allows silent numeric overflow when you call this function

on numeric data types. For more information on this behavior and how to change it, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

Example

The following query returns the total sum of the `product_cost` column.

```
=> SELECT SUM(product_cost) AS cost FROM product_dimension;
      cost
-----
  9042850
(1 row)
```

See Also

- [AVG \[Aggregate\]](#)
- [COUNT \[Aggregate\]](#)

SUM_FLOAT [Aggregate]

Computes the sum of an expression over a group of rows and returns a `DOUBLE PRECISION` value.

Behavior Type

Immutable

Syntax

```
SUM_FLOAT ( [ ALL | DISTINCT ] expression )
```

Parameters

ALL	Invokes the aggregate function for all rows in the group (default).
DISTINCT	Invokes the aggregate function for all distinct non-null values of the expression found in the group.

<i>expression</i>	Any expression whose result is type DOUBLE PRECISION.
-------------------	---

Overflow Handling

By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

Example

The following query returns the floating-point sum of the average price from the product table:

```
=> SELECT SUM_FLOAT(average_competitor_price) AS cost FROM product_dimension;
      cost
-----
 18181102
(1 row)
```

VAR_POP [Aggregate]

Evaluates the population variance for each member of the group. This is defined as the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of remaining rows:

$$\frac{(\text{SUM}(\text{expression} * \text{expression}) - \text{SUM}(\text{expression}) * \text{SUM}(\text{expression}) / \text{COUNT}(\text{expression}))}{\text{COUNT}(\text{expression})}$$

Behavior Type

Immutable

Syntax

```
VAR_POP ( expression )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. VAR_POP returns the same
-------------------	--

	data type as <i>expression</i> .
--	----------------------------------

Related Functions

This aggregate function differs from analytic function [VAR_POP](#), which computes the population variance of the current row with respect to the group of rows within a window.

Examples

The following example returns the population variance for each household ID in the customer table.

```
=> SELECT VAR_POP(household_id) FROM customer_dimension;
      var_pop
-----
74847050.0168393
(1 row)
```

VAR_SAMP [Aggregate]

Evaluates the sample variance for each row of the group. This is defined as the sum of squares of the difference of *expression* from the mean of *expression* divided by the number of remaining rows minus 1:

$$\frac{(\text{SUM}(\textit{expression} * \textit{expression}) - \text{SUM}(\textit{expression}) * \text{SUM}(\textit{expression}) / \text{COUNT}(\textit{expression}))}{(\text{COUNT}(\textit{expression}) - 1)}$$

Behavior Type

Immutable

Syntax

```
VAR_SAMP ( expression )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be
-------------------	--

implicitly converted to a numeric data type. VAR_SAMP returns the same data type as *expression*.

Related Functions

- VAR_SAMP is semantically identical to nonstandard function [VARIANCE](#), which is provided for compatibility with other databases.
- This aggregate function differs from analytic function [VAR_SAMP](#), which computes the sample variance of the current row with respect to the group of rows within a window.

Examples

The following example returns the sample variance for each household ID in the customer table.

```
=> SELECT VAR_SAMP(household_id) FROM customer_dimension;
      var_samp
-----
74848598.0106764
(1 row)
```

See Also

[VARIANCE \[Aggregate\]](#)

VARIANCE [Aggregate]

Evaluates the sample variance for each row of the group. This is defined as the sum of squares of the difference of *expression* from the mean of *expression* divided by the number of remaining rows minus 1.

```
(SUM(expression*expression) - SUM(expression) *SUM(expression) /COUNT(expression)) / (COUNT(expression) -1)
```

Behavior Type

Immutable

Syntax

VARIANCE (*expression*)

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. VARIANCE returns the same data type as <i>expression</i> .
-------------------	--

Related Functions

The nonstandard function VARIANCE is provided for compatibility with other databases. It is semantically identical to [VAR_SAMP](#).

This aggregate function differs from analytic function [VARIANCE](#), which computes the sample variance of the current row with respect to the group of rows within a window.

Examples

The following example returns the sample variance for each household ID in the customer table.

```
=> SELECT VARIANCE(household_id) FROM customer_dimension;
      variance
-----
74848598.0106764
(1 row)
```

See Also

- [Analytic Functions](#)
- [VAR_SAMP \[Aggregate\]](#)
- [SQL Analytics](#)

Analytic Functions

Note: All analytic functions in this section with an aggregate counterpart are appended with [Analytics] in the heading to avoid confusion between the two function types.

Vertica analytics are SQL functions based on the ANSI 99 standard. These functions handle complex analysis and reporting tasks—for example:

- Rank the longest-standing customers in a particular state.
- Calculate the moving average of retail volume over a specified time.
- Find the highest score among all students in the same grade.
- Compare the current sales bonus that salespersons received against their previous bonus.

Analytic functions return aggregate results but they do not group the result set. They return the group value multiple times, once per record. You can sort group values, or partitions, using a window `ORDER BY` clause, but the order affects only the function result set, not the entire query result set.

Syntax

General

```
analytic-function ( arguments ) OVER(  
...[ window-partition-clause ]  
...[ window-order-clause [ window-frame-clause ] ]  
)
```

With named window

```
analytic-function ( arguments ) OVER(  
...[ named-window [ window-frame-clause ] ]  
)
```

Parameters

<i>analytic-function</i> (<i>arguments</i>)	A Vertica analytic function and its arguments.
--	--

OVER	<p>Specifies how to partition, sort, and window frame function input with respect to the current row. The input data is the result set that the query returns after it evaluates FROM, WHERE, GROUP BY, and HAVING clauses.</p> <p>An empty OVER clause provides the best performance for single threaded queries on a single node.</p>
<i>window-partition-clause</i>	<p>Groups input rows according to one or more columns or expressions.</p> <p>If you omit this clause, no grouping occurs and the analytic function processes all input rows as a single partition.</p>
<i>window-order-clause</i>	<p>Optionally specifies how to sort rows that are supplied to the analytic function. If the OVER clause also includes a partition clause, rows are sorted within each partition.</p>
<i>window-frame-clause</i>	<p>Only valid for some analytic functions, specifies as input a set of rows relative to the row that is currently being evaluated by the analytic function. After the function processes that row and its window, Vertica advances the current row and adjusts the window boundaries accordingly.</p>
<i>named-window</i>	<p>The name of a window that you define in the same query with a window name clause. This definition encapsulates window partitioning and sorting. Named windows are useful when the query invokes multiple analytic functions with similar OVER clauses.</p> <p>A window name clause cannot specify a window frame clause. However, you can qualify the named window in an OVER clause with a window frame clause.</p>

Requirements

The following requirements apply to analytic functions:

- All require an OVER clause. Each function has its own OVER clause requirements. For example, you can supply an empty OVER clause for some analytic aggregate functions such as [SUM](#). For other functions, window frame and order clauses might be required, or might be invalid.
- Analytic functions can be invoked only in a query's SELECT and ORDER BY clauses.
- Analytic functions cannot be nested. For example, the following query is not allowed:

```
=> SELECT MEDIAN(RANK() OVER(ORDER BY sal) OVER()).
```

- WHERE, GROUP BY and HAVING operators are technically not part of the analytic function. However, they determine input to that function.

See Also

- [SQL Analytics](#)
- [GROUP BY Queries](#)

Window Partition Clause

When specified, a window partition clause divides the rows of the function input based on user-provided expressions. If no expression is provided, the partition clause can improve query performance by using parallelism.

Window partitioning is similar to the GROUP BY clause except that it returns only one result row per input row. If you omit specifying a window partition clause, all input rows are treated as a single partition.

When used with analytic functions, results are computed per partition and start over again (reset) at the beginning of each subsequent partition.

Syntax

```
{ PARTITION BY expression[,...] | PARTITION BEST | PARTITION NODES }
```

Parameters

<code>PARTITION BY <i>expression</i></code>	Expression on which to sort the partition, where <i>expression</i> can be a column, constant, or an arbitrary expression formed on columns. Use <code>PARTITION BY</code> for analytic functions with specific partitioning requirements.
<code>PARTITION BEST</code>	<p>Use parallelism to improve performance for multi-threaded queries across multiple nodes.</p> <p><code>OVER(PARTITION BEST)</code> provides the best performance on multi-threaded queries across multiple nodes.</p> <p>The following considerations apply to using <code>PARTITION BEST</code>:</p> <ul style="list-style-type: none">• Use <code>PARTITION BEST</code> for analytic functions that have no partitioning requirements and are thread safe—for example, a one-to-many transform.• Do not use <code>PARTITION BEST</code> on user-defined transform functions (UDTFs) that are not thread-safe. Doing so can produce an error or incorrect results. If a UDTF is not thread safe, use <code>PARTITION NODES</code>.
<code>PARTITION NODES</code>	<p>Use parallelism to improve performance for single-threaded queries across multiple nodes.</p> <p><code>OVER(PARTITION NODES)</code> provides the best performance on single-threaded queries across multiple nodes.</p>

Examples

See [Window Partitioning](#) in Analyzing Data.

Window Order Cause

Specifies how to sort rows that are supplied to the analytic function. If the `OVER` clause also includes a [window partition clause](#), rows are sorted within each partition.

The window order clause only specifies order within a window result set. The query can have its own **ORDER BY** clause outside the OVER clause. This has precedence over the window order clause and orders the final result set.

An window order clause also creates a default **window frame** if none is explicitly specified.

Syntax

```
ORDER BY { expression[ sort-qualifiers ] } [,...]
```

sort-qualifiers =

```
[ ASC | DESC ]  
[ NULLS { FIRST | LAST | AUTO } ]
```

Parameters

<i>expression</i>	A column, constant, or arbitrary expression formed on columns, on which to sort input rows.
ASC DESC	Specifies the ordering sequence as ascending (default) or descending.
NULLS {FIRST LAST AUTO}	<p>Specifies whether to position null values first or last. Default positioning depends on whether the sort order is ascending or descending:</p> <ul style="list-style-type: none">• Ascending default: NULLS LAST• Descending default: NULLS FIRST <p>If you specify NULLS AUTO, Vertica chooses the positioning that is most efficient for this query, either either NULLS FIRST or NULLS LAST.</p> <p>If you omit all sort qualifiers, Vertica uses ASC NULLS LAST.</p> <p>For more information, see:</p> <ul style="list-style-type: none">• NULL Sort Order• Runtime Sorting of NULL Values in Analytic Functions

Examples

See [Window Ordering](#) in Analyzing Data.

Window Frame Clause

Specifies a window frame, which comprises a set of rows relative to the row that is currently being evaluated by the analytic function. After the function processes that row and its window, Vertica advances the current row and adjusts the window boundaries accordingly. If the OVER clause also specifies a [partition](#), Vertica also checks that window boundaries do not cross partition boundaries. This process repeats until the function evaluates the last row of the last partition.

Syntax

```
{ ROWS | RANGE }  
  { BETWEEN start-point AND end-point } | start-point  
  
start-point / end-point =  
{ UNBOUNDED {PRECEDING | FOLLOWING}  
  | CURRENT ROW  
  | constant-value {PRECEDING | FOLLOWING}  
}
```

Parameters

ROWS RANGE	Specifies whether Vertica determines window frame dimensions as physical or logical offsets from the current row. See ROWS versus RANGE below for details.
BETWEEN <i>start-point</i> AND <i>end-point</i>	Specifies the window's first and last rows, where <i>start-point</i> and <i>end-point</i> can be one of the following (discussed in detail below): <ul style="list-style-type: none">• UNBOUNDED {PRECEDING FOLLOWING}• CURRENT ROW

	<ul style="list-style-type: none"> • <i>constant-value</i> {PRECEDING FOLLOWING} <p><i>start-point</i> must resolve to a row or value that is less than or equal to <i>end-point</i>.</p>
UNBOUNDED PRECEDING	Specifies that the window frame extends to the current partition's first row.
<i>start-point</i>	If ROWS or RANGE specifies only a start point, Vertica uses the current row as the end point and creates the window frame accordingly. In this case, <i>start-point</i> must resolve to a row that is less than or equal to the current row.
UNBOUNDED FOLLOWING	Specifies that the window frame extends to the current partition's last row.
CURRENT ROW	Specifies the current row or value as the window's start or end point.
<i>constant-value</i> {PRECEDING FOLLOWING}	<p>Specifies a constant value or expression that evaluates to a constant value. The value specifies a physical or logical offset from the current row, depending on whether you specify ROWS or RANGE.</p> <p>Other dependencies also pertain, depending whether you specify ROWS and RANGE. See ROWS versus RANGE below for details.</p>

Requirements

In order to specify a window frame, the OVER must also specify a [window order](#) (ORDER BY) [clause](#). If the OVER clause omits specifying a window frame, the function creates a default window that extends from the current row to the first row in the current partition. This is equivalent to the following clause:

```
RANGE UNBOUNDED PRECEDING AND CURRENT ROW
```

ROWS versus RANGE

The window frame's offset from the current row can be physical or logical:

- ROWS specifies the window's *start-point* and *end-point* as a number of rows relative to the current row. If *start-point* and *end-point* are expressed as constant values, the value must evaluate to a positive integer.
- RANGE specifies the window as a logical offset such as time. The range value must match the [window order](#) (ORDER BY) [clause](#) data type: NUMERIC, DATE/TIME, FLOAT or INTEGER.

Use of ROWS or RANGE imposes specific requirements on setting the window's start and end points as constant values:

Setting constant values for ROWS

The constant must evaluate to a positive INTEGER.

Setting constant values for RANGE

The following requirements apply:

- The constant must evaluate to a positive numeric value or INTERVAL literal.
- If the constant evaluates to a NUMERIC value, the ORDER BY column type must be a NUMERIC data type.
- If the constant evaluates to an INTERVAL DAY TO SECOND subtype, the ORDER BY column type must be one of the following: TIMESTAMP, TIME, DATE, or INTERVAL DAY TO SECOND.
- If the constant evaluates to an INTERVAL YEAR TO MONTH, the ORDER BY column type must be one of the following: TIMESTAMP, DATE, or INTERVAL YEAR TO MONTH.
- The [window order clause](#) can specify only one expression.

Examples

See [Window Framing](#) in Analyzing Data

Window Name Clause

Defines a named window that specifies window partition and order clauses for an analytic function. This window is specified in the function's OVER clause. Named windows can be useful when you write queries that invoke multiple analytic functions with similar OVER clauses—for example, they use the same partition (PARTITION BY) clauses.

Syntax

WINDOW *window-name* AS (*window-partition-clause* [*window-order-clause*])

Parameters

WINDOW <i>window-name</i>	Specifies the window name. All window names must be unique within the same query.
<i>window-partition-clause</i> [<i>window-order-clause</i>]	Clauses to invoke when an OVER clause references this window. If the window definition omits a window order clause , the OVER clause can specify its own order clause.

Requirements

- A WINDOW clause cannot include a [window frame clause](#).
- Each WINDOW clause within the same query must have a unique name.
- A WINDOW clause can reference another window that is already named. For example, the following query names window w1 before w2. Thus, the WINDOW clause that defines w2 can reference w1:

```
=> SELECT RANK() OVER(w1 ORDER BY sal DESC), RANK() OVER w2  
FROM EMP WINDOW w1 AS (PARTITION BY deptno), w2 AS (w1 ORDER BY sal);
```

Examples

See [Named Windows](#) in Analyzing Data.

See Also

[Analytic Functions](#)

AVG [Analytic]

Computes an average of an expression in a group within a window. AVG returns the same data type as the expression's numeric data type.

The AVG analytic function differs from the [AVG](#) aggregate function, which computes the average of an expression over a group of rows.

Behavior Type

Immutable

Syntax

```
AVG ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any data that can be implicitly converted to a numeric data type.
OVER()	See Analytic Functions .

Overflow Handling

By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

Examples

The following query finds the sales for that calendar month and returns a running/cumulative average (sometimes called a moving average) using the default window of RANGE UNBOUNDED PRECEDING AND CURRENT ROW:

```
=> SELECT calendar_month_number_in_year Mo, SUM(product_price) Sales,
        AVG(SUM(product_price)) OVER (ORDER BY calendar_month_number_in_year)::INTEGER Average
        FROM product_dimension pd, date_dimension dm, inventory_fact if
        WHERE dm.date_key = if.date_key AND pd.product_key = if.product_key GROUP BY Mo;
```

Mo	Sales	Average
1	23869547	23869547
2	19604661	21737104
3	22877913	22117374
4	22901263	22313346
5	23670676	22584812
6	22507600	22571943
7	21514089	22420821
8	24860684	22725804
9	21687795	22610470
10	23648921	22714315
11	21115910	22569005
12	24708317	22747281

(12 rows)

To return a moving average that is not a running (cumulative) average, the window can specify **ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING**:

```
=> SELECT calendar_month_number_in_year Mo, SUM(product_price) Sales,
        AVG(SUM(product_price)) OVER (ORDER BY calendar_month_number_in_year
        ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)::INTEGER Average
        FROM product_dimension pd, date_dimension dm, inventory_fact if
        WHERE dm.date_key = if.date_key AND pd.product_key = if.product_key GROUP BY Mo;
```

Mo	Sales	Average
1	23869547	22117374
2	19604661	22313346
3	22877913	22584812
4	22901263	22312423
5	23670676	22694308
6	22507600	23090862
7	21514089	22848169
8	24860684	22843818
9	21687795	22565480
10	23648921	23204325
11	21115910	22790236
12	24708317	23157716

(12 rows)

See Also

- [COUNT \[Analytic\]](#)
- [SUM \[Analytic\]](#)
- [SQL Analytics](#)

BOOL_AND [Analytic]

Returns the Boolean value of an expression within a window. If all input values are true, BOOL_AND returns t. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

```
BOOL_AND ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	A Boolean Data Type or any non-Boolean data type that can be implicitly converted to a Boolean data type. The function returns a Boolean value.
OVER()	See Analytic Functions .

Examples

The following example illustrates how you can use the BOOL_AND, BOOL_OR, and BOOL_XOR analytic functions. The sample table, employee, includes a column for type of employee and years paid.

```
=> CREATE TABLE employee(emptytype VARCHAR, yearspaid VARCHAR);
CREATE TABLE
```

Insert sample data into the table to show years paid. In more than one case, an employee could be paid more than once within one year.

```
=> INSERT INTO employee
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2015'
```

```
UNION ALL
SELECT 'contractor3', '2014'
UNION ALL
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2014'
UNION ALL
SELECT 'contractor3', '2015'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor5', '2015'
UNION ALL
SELECT 'contractor5', '2016';
OUTPUT
-----
      10
(1 row)
```

Query the table. The result shows employees that were paid twice in 2014 (BOOL_AND), once or twice in 2014 (BOOL_OR), and specifically not more than once in 2014 (BOOL_XOR).

```
=> SELECT DISTINCT emptype,
BOOL_AND(yearspaid='2014') OVER (PARTITION BY emptype) AS paidtwicein2014,
BOOL_OR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidonceortwicein2014,
BOOL_XOR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidjustoncein2014
FROM employee;
```

emptype	paidtwicein2014	paidonceortwicein2014	paidjustoncein2014
contractor1	t	t	f
contractor2	f	t	t
contractor3	f	t	t
contractor4	t	t	f
contractor5	f	f	f

(5 rows)

See Also

- [BOOL_AND \[Aggregate\]](#)
- [BOOL_OR \[Analytic\]](#)
- [BOOL_XOR \[Analytic\]](#)
- [Boolean Data Type](#)

BOOL_OR [Analytic]

Returns the Boolean value of an expression within a window. If at least one input value is true, BOOL_OR returns t. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

```
BOOL_OR ( expression ) OVER (  
... [ window-partition-clause ]  
... [ window-order-clause ]  
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	A Boolean Data Type or any non-Boolean data type that can be implicitly converted to a Boolean data type. The function returns a Boolean value.
OVER()	See Analytic Functions .

Examples

The following example illustrates how you can use the BOOL_AND, BOOL_OR, and BOOL_XOR analytic functions. The sample table, employee, includes a column for type of employee and years paid.

```
=> CREATE TABLE employee(emptytype VARCHAR, yearspaid VARCHAR);  
CREATE TABLE
```

Insert sample data into the table to show years paid. In more than one case, an employee could be paid more than once within one year.

```
=> INSERT INTO employee  
SELECT 'contractor1', '2014'  
UNION ALL  
SELECT 'contractor2', '2015'
```

```
UNION ALL
SELECT 'contractor3', '2014'
UNION ALL
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2014'
UNION ALL
SELECT 'contractor3', '2015'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor5', '2015'
UNION ALL
SELECT 'contractor5', '2016';
OUTPUT
-----
      10
(1 row)
```

Query the table. The result shows employees that were paid twice in 2014 (BOOL_AND), once or twice in 2014 (BOOL_OR), and specifically not more than once in 2014 (BOOL_XOR).

```
=> SELECT DISTINCT emptype,
BOOL_AND(yearspaid='2014') OVER (PARTITION BY emptype) AS paidtwicein2014,
BOOL_OR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidonceortwicein2014,
BOOL_XOR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidjustoncein2014
FROM employee;
```

emptype	paidtwicein2014	paidonceortwicein2014	paidjustoncein2014
contractor1	t	t	f
contractor2	f	t	t
contractor3	f	t	t
contractor4	t	t	f
contractor5	f	f	f

(5 rows)

See Also

- [BOOL_OR \[Aggregate\]](#)
- [BOOL_AND \[Analytic\]](#)
- [BOOL_XOR \[Analytic\]](#)
- [Boolean Data Type](#)

BOOL_XOR [Analytic]

Returns the Boolean value of an expression within a window. If only one input value is true, BOOL_XOR returns t. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

```
BOOL_XOR ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	A Boolean Data Type or any non-Boolean data type that can be implicitly converted to a Boolean data type. The function returns a Boolean value.
OVER()	See Analytic Functions .

Examples

The following example illustrates how you can use the BOOL_AND, BOOL_OR, and BOOL_XOR analytic functions. The sample table, employee, includes a column for type of employee and years paid.

```
=> CREATE TABLE employee(emptytype VARCHAR, yearspaid VARCHAR);
CREATE TABLE
```

Insert sample data into the table to show years paid. In more than one case, an employee could be paid more than once within one year.

```
=> INSERT INTO employee
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2015'
```

```
UNION ALL
SELECT 'contractor3', '2014'
UNION ALL
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2014'
UNION ALL
SELECT 'contractor3', '2015'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor5', '2015'
UNION ALL
SELECT 'contractor5', '2016';
OUTPUT
-----
      10
(1 row)
```

Query the table. The result shows employees that were paid twice in 2014 (BOOL_AND), once or twice in 2014 (BOOL_OR), and specifically not more than once in 2014 (BOOL_XOR).

```
=> SELECT DISTINCT emptype,
BOOL_AND(yearspaid='2014') OVER (PARTITION BY emptype) AS paidtwicein2014,
BOOL_OR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidonceortwicein2014,
BOOL_XOR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidjustoncein2014
FROM employee;
```

emptype	paidtwicein2014	paidonceortwicein2014	paidjustoncein2014
contractor1	t	t	f
contractor2	f	t	t
contractor3	f	t	t
contractor4	t	t	f
contractor5	f	f	f

(5 rows)

See Also

- [BOOL_XOR \[Aggregate\]](#)
- [BOOL_AND \[Analytic\]](#)
- [BOOL_OR \[Analytic\]](#)
- [Boolean Data Type](#)

CONDITIONAL_CHANGE_EVENT [Analytic]

Assigns an event window number to each row, starting from 0, and increments by 1 when the result of evaluating the argument expression on the current row differs from that on the previous row.

Behavior Type

Immutable

Syntax

```
CONDITIONAL_CHANGE_EVENT ( expression ) OVER (
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

<i>expression</i>	SQL scalar expression that is evaluated on an input record. The result of <i>expression</i> can be of any data type.
OVER()	See Analytic Functions .

Notes

The analytic *window-order-clause* is required but the *window-partition-clause* is optional.

Example

```
=> SELECT CONDITIONAL_CHANGE_EVENT(bid)
      OVER (PARTITION BY symbol ORDER BY ts) AS cce
FROM TickStore;
```

The system returns an error when no ORDER BY clause is present:

```
=> SELECT CONDITIONAL_CHANGE_EVENT(bid)
      OVER (PARTITION BY symbol) AS cce
   FROM TickStore;

ERROR: conditional_change_event must contain an
ORDER BY clause within its analytic clause
```

For more examples, see [Event-Based Windows](#) in Analyzing Data.

See Also

- [CONDITIONAL_TRUE_EVENT \[Analytic\]](#)
- [ROW_NUMBER \[Analytic\]](#)
- [Time Series Analytics](#)
- [Event-Based Windows](#)

CONDITIONAL_TRUE_EVENT [Analytic]

Assigns an event window number to each row, starting from 0, and increments the number by 1 when the result of the boolean argument expression evaluates true. For example, given a sequence of values for column *a*, as follows:

```
( 1, 2, 3, 4, 5, 6 )
```

`CONDITIONAL_TRUE_EVENT(a > 3)` returns 0, 0, 0, 1, 2, 3.

Behavior Type:

Immutable

Syntax

```
CONDITIONAL_TRUE_EVENT ( boolean-expression ) OVER
... ( [ window-partition-clause ]
... window-order-clause )
```

Parameters

<i>boolean-expression</i>	SQL scalar expression that is evaluated on an input record, type BOOLEAN.
OVER()	See Analytic Functions .

Notes

The analytic *window-order-clause* is required but the *window-partition-clause* is optional.

Example

```
> SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6)
   OVER(PARTITION BY bid ORDER BY ts) AS cte
FROM Tickstore;
```

The system returns an error if the ORDER BY clause is omitted:

```
> SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6)
   OVER(PARTITION BY bid) AS cte
FROM Tickstore;

ERROR: conditional_true_event must contain an ORDER BY
clause within its analytic clause
```

For more examples, see [Event-Based Windows](#) in Analyzing Data.

See Also

- [CONDITIONAL_CHANGE_EVENT \[Analytic\]](#)
- [Time Series Analytics](#)
- [Event-Based Windows](#)

COUNT [Analytic]

Counts occurrences within a group within a window. If you specify * or some non-null constant, COUNT() counts all rows.

Behavior Type

Immutable

Syntax

```
COUNT ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Returns the number of rows in each group for which the <i>expression</i> is not null. Can be any expression resulting in BIGINT.
OVER()	See Analytic Functions .

Example

Using the schema defined in [Window Framing](#) in Analyzing Data, the following COUNT function omits window order and window frame clauses; otherwise Vertica would treat it as a window aggregate. Think of the window of reporting aggregates as UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING.

```
=> SELECT deptno, sal, empno, COUNT(sal)
      OVER (PARTITION BY deptno) AS count FROM emp;
```

```
deptno | sal | empno | count
-----+-----+-----+-----
    10 | 101 |     1 |     2
    10 | 104 |     4 |     2
    20 | 110 |    10 |     6
    20 | 110 |     9 |     6
    20 | 109 |     7 |     6
```

20	109	6	6
20	109	8	6
20	109	11	6
30	105	5	3
30	103	3	3
30	102	2	3

Using `ORDER BY sal` creates a moving window query with default window: `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.

```
=> SELECT deptno, sal, empno, COUNT(sal)
      OVER (PARTITION BY deptno ORDER BY sal) AS count
      FROM emp;
```

deptno	sal	empno	count
10	101	1	1
10	104	4	2
20	100	11	1
20	109	7	4
20	109	6	4
20	109	8	4
20	110	10	6
20	110	9	6
30	102	2	1
30	103	3	2
30	105	5	3

Using the VMart schema, the following query finds the number of employees who make less than or equivalent to the hourly rate of the current employee. The query returns a running/cumulative average (sometimes called a moving average) using the default window of `RANGE UNBOUNDED PRECEDING AND CURRENT ROW`:

```
=> SELECT employee_last_name AS "last_name", hourly_rate, COUNT(*)
      OVER (ORDER BY hourly_rate) AS moving_count from employee_dimension;
```

last_name	hourly_rate	moving_count
Gauthier	6	4
Taylor	6	4
Jefferson	6	4
Nielson	6	4
McNulty	6.01	11
Robinson	6.01	11
Dobisz	6.01	11
Williams	6.01	11
Kramer	6.01	11
Miller	6.01	11
Wilson	6.01	11
Vogel	6.02	14
Moore	6.02	14
Vogel	6.02	14
Carcetti	6.03	19
...		

To return a moving average that is not also a running (cumulative) average, the window should specify `ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING`:

```
=> SELECT employee_last_name AS "last_name", hourly_rate, COUNT(*)  
       OVER (ORDER BY hourly_rate ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)  
       AS moving_count from employee_dimension;
```

See Also

- [COUNT \[Aggregate\]](#)
- [AVG \[Analytic\]](#)
- [SUM \[Analytic\]](#)
- [SQL Analytics](#)

CUME_DIST [Analytic]

Calculates the cumulative distribution, or relative rank, of the current row with regard to other rows in the same partition within a window.

`CUME_DIST()` returns a number greater than 0 and less than or equal to 1, where the number represents the relative position of the specified row within a group of n rows. For a row x (assuming `ASC` ordering), the `CUME_DIST` of x is the number of rows with values lower than or equal to the value of x , divided by the number of rows in the partition. For example, in a group of three rows, the cumulative distribution values returned would be $1/3$, $2/3$, and $3/3$.

Note: Because the result for a given row depends on the number of rows preceding that row in the same partition, you should always specify a *window-order-clause* when you call this function.

Behavior Type

Immutable

Syntax

```
CUME_DIST ( ) OVER (  
... [ window-partition-clause ]  
... window-order-clause )
```

Parameters

OVER()	See Analytic Functions .
--------	--

Examples

The following example returns the cumulative distribution of sales for different transaction types within each month of the first quarter.

```
=> SELECT calendar_month_name AS month, tender_type, SUM(sales_quantity),
        CUME_DIST()
        OVER (PARTITION BY calendar_month_name ORDER BY SUM(sales_quantity)) AS
CUME_DIST
FROM store.store_sales_fact JOIN date_dimension
USING(date_key) WHERE calendar_month_name IN ('January','February','March')
AND tender_type NOT LIKE 'Other'
GROUP BY calendar_month_name, tender_type;
```

month	tender_type	SUM	CUME_DIST
March	Credit	469858	0.25
March	Cash	470449	0.5
March	Check	473033	0.75
March	Debit	475103	1
January	Cash	441730	0.25
January	Debit	443922	0.5
January	Check	446297	0.75
January	Credit	450994	1
February	Check	425665	0.25
February	Debit	426726	0.5
February	Credit	430010	0.75
February	Cash	430767	1

(12 rows)

See Also

- [PERCENT_RANK \[Analytic\]](#)
- [PERCENTILE_DISC \[Analytic\]](#)
- [SQL Analytics](#)

DENSE_RANK [Analytic]

Within each window partition, ranks all rows in the query results set according to the order specified by the window's `ORDER BY` clause. A `DENSE_RANK` function returns a sequence of ranking numbers without any gaps.

`DENSE_RANK` executes as follows:

1. Sorts partition rows as specified by the `ORDER BY` clause.
 2. Compares the `ORDER BY` values of the preceding row and current row and ranks the current row as follows:
 - If `ORDER BY` values are the same, the current row gets the same ranking as the preceding row.
- Note:** Null values are considered equal. For detailed information on how null values are sorted, see [NULL Sort Order](#).
- If the `ORDER BY` values are different, `DENSE_RANK` increments or decrements the current row's ranking by 1, depending whether sort order is ascending or descending.

`DENSE_RANK` always changes the ranking by 1, so no gaps appear in the ranking sequence. The largest rank value is the number of unique `ORDER BY` values returned by the query.

Behavior Type

Immutable

Syntax

```
DENSE_RANK() OVER(  
... [ window-partition-clause ]  
... window-order-clause )
```

Parameters

OVER()	See Analytic Functions .
--------	--

See [Analytic Functions](#)

Compared with RANK

RANK leaves gaps in the ranking sequence, while **DENSE_RANK** does not. The example below compares the behavior of the two functions.

Example

The following query invokes **RANK** and **DENSE_RANK** to rank customers by annual income. The two functions return different rankings, as follows:

- If `annual_salary` contains duplicate values, **RANK()** inserts duplicate rankings and then skips one or more values—for example, from 4 to 6 and 7 to 9.
- In the parallel column `Dense Rank`, **DENSE_RANK()** also inserts duplicate rankings, but leaves no gaps in the rankings sequence:

```
=> SELECT employee_region region, employee_key, annual_salary,
       RANK() OVER (PARTITION BY employee_region ORDER BY annual_salary) Rank,
       DENSE_RANK() OVER (PARTITION BY employee_region ORDER BY annual_salary) "Dense Rank"
FROM employee_dimension;
```

region	employee_key	annual_salary	Rank	Dense Rank
West	5248	1200	1	1
West	6880	1204	2	2
West	5700	1214	3	3
West	9857	1218	4	4
West	6014	1218	4	4
West	9221	1220	6	5
West	7646	1222	7	6
West	6621	1222	7	6
West	6488	1224	9	7
West	7659	1226	10	8
West	7432	1226	10	8
West	9905	1226	10	8
West	9021	1228	13	9
...				
West	56	963104	2794	2152
West	100	992363	2795	2153
East	8353	1200	1	1
East	9743	1202	2	2
East	9975	1202	2	2
East	9205	1204	4	3
East	8894	1206	5	4
East	7740	1206	5	4
East	7324	1208	7	5
East	6505	1208	7	5
East	5404	1208	7	5
East	5010	1208	7	5
East	9114	1212	11	6
...				

See Also

[SQL Analytics](#)

EXPONENTIAL_MOVING_AVERAGE [Analytic]

Calculates the exponential moving average (EMA) of expression E with smoothing factor X . An EMA differs from a simple moving average in that it provides a more stable picture of changes to data over time.

The EMA is calculated by adding the previous EMA value to the current data point scaled by the smoothing factor, as in the following formula:

$$EMA = EMA\theta + (X * (E - EMA\theta))$$

where:

- E is the current data point
- $EMA\theta$ is the previous row's EMA value.
- X is the smoothing factor.

This function also works at the row level. For example, EMA assumes the data in a given column is sampled at uniform intervals. If the users' data points are sampled at non-uniform intervals, they should run the time series [gap filling and interpolation \(GFI\)](#) operations before `EMA()`

Behavior Type

Immutable

Syntax

```
EXPONENTIAL_MOVING_AVERAGE (  $E$ ,  $X$  ) OVER (
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

<i>E</i>	The value whose average is calculated over a set of rows. Can be INTEGER, FLOAT or NUMERIC type and must be a constant.
<i>X</i>	A positive FLOAT value between 0 and 1 that is used as the smoothing factor.
OVER ()	See Analytic Functions .

Examples

The following example uses time series [gap filling and interpolation](#) (GFI) first in a subquery, and then performs an EXPONENTIAL_MOVING_AVERAGE operation on the subquery result.

Create a simple four-column table:

```
=> CREATE TABLE ticker(  
    time TIMESTAMP,  
    symbol VARCHAR(8),  
    bid1 FLOAT,  
    bid2 FLOAT );
```

Insert some data, including nulls, so GFI can do its interpolation and gap filling:

```
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:00', 'ABC', 60.45, 60.44);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:01', 'ABC', 60.49, 65.12);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:02', 'ABC', 57.78, 59.25);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:03', 'ABC', null, 65.12);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:04', 'ABC', 67.88, null);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:00', 'XYZ', 47.55, 40.15);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:01', 'XYZ', 44.35, 46.78);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:02', 'XYZ', 71.56, 75.78);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:03', 'XYZ', 85.55, 70.21);  
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:04', 'XYZ', 45.55, 58.65);  
=> COMMIT;
```

Note: During gap filling and interpolation, Vertica takes the closest non null value on either side of the time slice and uses that value. For example, if you use a linear interpolation scheme and you do not specify IGNORE NULLS, and your data has one real value and one null, the result is null. If the value on either side is null, the result is null. See [When Time Series Data Contains Nulls](#) in Analyzing Data for details.

Query the table that you just created to you can see the output:

```
=> SELECT * FROM ticker;
      time          | symbol | bid1 | bid2
-----+-----+-----+-----
2009-07-12 03:00:00 | ABC    | 60.45 | 60.44
2009-07-12 03:00:01 | ABC    | 60.49 | 65.12
2009-07-12 03:00:02 | ABC    | 57.78 | 59.25
2009-07-12 03:00:03 | ABC    |      | 65.12
2009-07-12 03:00:04 | ABC    | 67.88 |
2009-07-12 03:00:00 | XYZ    | 47.55 | 40.15
2009-07-12 03:00:01 | XYZ    | 44.35 | 46.78
2009-07-12 03:00:02 | XYZ    | 71.56 | 75.78
2009-07-12 03:00:03 | XYZ    | 85.55 | 70.21
2009-07-12 03:00:04 | XYZ    | 45.55 | 58.65
(10 rows)
```

The following query processes the first and last values that belong to each 2-second time slice in table `trades'` column `a`. The query then calculates the exponential moving average of expression `fv` and `lv` with a smoothing factor of 50%:

```
=> SELECT symbol, slice_time, fv, lv,
      EXPONENTIAL_MOVING_AVERAGE(fv, 0.5)
      OVER (PARTITION BY symbol ORDER BY slice_time) AS ema_first,
      EXPONENTIAL_MOVING_AVERAGE(lv, 0.5)
      OVER (PARTITION BY symbol ORDER BY slice_time) AS ema_last
FROM (
  SELECT symbol, slice_time,
         TS_FIRST_VALUE(bid1 IGNORE NULLS) as fv,
         TS_LAST_VALUE(bid2 IGNORE NULLS) AS lv
  FROM ticker TIMESERIES slice_time AS '2 seconds'
  OVER (PARTITION BY symbol ORDER BY time) ) AS sq;
```

```
symbol | slice_time          | fv   | lv   | ema_first | ema_last
-----+-----+-----+-----+-----+-----
ABC    | 2009-07-12 03:00:00 | 60.45 | 65.12 | 60.45    | 65.12
ABC    | 2009-07-12 03:00:02 | 57.78 | 65.12 | 59.115   | 65.12
ABC    | 2009-07-12 03:00:04 | 67.88 | 65.12 | 63.4975  | 65.12
XYZ    | 2009-07-12 03:00:00 | 47.55 | 46.78 | 47.55    | 46.78
XYZ    | 2009-07-12 03:00:02 | 71.56 | 70.21 | 59.555   | 58.495
XYZ    | 2009-07-12 03:00:04 | 45.55 | 58.65 | 52.5525  | 58.5725
(6 rows)
```

See Also

- [TIMESERIES Clause](#)
- [Time Series Analytics](#)
- [SQL Analytics](#)

FIRST_VALUE [Analytic]

Lets you select the first value of a table or partition (determined by the *window-order-clause*) without having to use a self join. This function is useful when you want to use the first value as a baseline in calculations.

Use `FIRST_VALUE()` with the *window-order-clause* to produce deterministic results. If no window is specified for the current row, the default window is `UNBOUNDED PRECEDING AND CURRENT ROW`.

Behavior Type

Immutable

Syntax

```
FIRST_VALUE ( expression [ IGNORE NULLS ] ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Expression to evaluate—or example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.
IGNORE NULLS	Specifies to return the first non-null value in the set, or NULL if all values are NULL. If you omit this option and the first value in the set is null, the function returns NULL.
OVER()	See Analytic Functions .

Examples

The following query asks for the first value in the partitioned day of week, and illustrates the potential nondeterministic nature of `FIRST_VALUE()`:

```
=> SELECT calendar_year, date_key, day_of_week, full_date_description,
       FIRST_VALUE(full_date_description)
       OVER(PARTITION BY calendar_month_number_in_year ORDER BY day_of_week)
       AS "first_value"
FROM date_dimension
WHERE calendar_year=2003 AND calendar_month_number_in_year=1;
```

The first value returned is January 31, 2003; however, the next time the same query is run, the first value might be January 24 or January 3, or the 10th or 17th. This is because the analytic `ORDER BY` column `day_of_week` returns rows that contain ties (multiple Fridays). These repeated values make the `ORDER BY` evaluation result nondeterministic, because rows that contain ties can be ordered in any way, and any one of those rows qualifies as being the first value of `day_of_week`.

calendar_year	date_key	day_of_week	full_date_description	first_value
2003	31	Friday	January 31, 2003	January 31, 2003
2003	24	Friday	January 24, 2003	January 31, 2003
2003	3	Friday	January 3, 2003	January 31, 2003
2003	10	Friday	January 10, 2003	January 31, 2003
2003	17	Friday	January 17, 2003	January 31, 2003
2003	6	Monday	January 6, 2003	January 31, 2003
2003	27	Monday	January 27, 2003	January 31, 2003
2003	13	Monday	January 13, 2003	January 31, 2003
2003	20	Monday	January 20, 2003	January 31, 2003
2003	11	Saturday	January 11, 2003	January 31, 2003
2003	18	Saturday	January 18, 2003	January 31, 2003
2003	25	Saturday	January 25, 2003	January 31, 2003
2003	4	Saturday	January 4, 2003	January 31, 2003
2003	12	Sunday	January 12, 2003	January 31, 2003
2003	26	Sunday	January 26, 2003	January 31, 2003
2003	5	Sunday	January 5, 2003	January 31, 2003
2003	19	Sunday	January 19, 2003	January 31, 2003
2003	23	Thursday	January 23, 2003	January 31, 2003
2003	2	Thursday	January 2, 2003	January 31, 2003
2003	9	Thursday	January 9, 2003	January 31, 2003
2003	16	Thursday	January 16, 2003	January 31, 2003
2003	30	Thursday	January 30, 2003	January 31, 2003
2003	21	Tuesday	January 21, 2003	January 31, 2003
2003	14	Tuesday	January 14, 2003	January 31, 2003
2003	7	Tuesday	January 7, 2003	January 31, 2003
2003	28	Tuesday	January 28, 2003	January 31, 2003
2003	22	Wednesday	January 22, 2003	January 31, 2003
2003	29	Wednesday	January 29, 2003	January 31, 2003
2003	15	Wednesday	January 15, 2003	January 31, 2003
2003	1	Wednesday	January 1, 2003	January 31, 2003
2003	8	Wednesday	January 8, 2003	January 31, 2003

(31 rows)

Note: The `day_of_week` results are returned in alphabetical order because of lexical rules. The fact that each day does not appear ordered by the 7-day week cycle (for example, starting with Sunday followed by Monday, Tuesday, and so on) has no affect on results.

To return deterministic results, modify the query so that it performs its analytic ORDER BY operations on a **unique** field, such as `date_key`:

```
=> SELECT calendar_year, date_key, day_of_week, full_date_description,
FIRST_VALUE(full_date_description) OVER
(PARTITION BY calendar_month_number_in_year ORDER BY date_key) AS "first_value"
FROM date_dimension WHERE calendar_year=2003;
```

`FIRST_VALUE()` returns a first value of January 1 for the January partition and the first value of February 1 for the February partition. Also, the `full_date_description` column contains no ties:

calendar_year	date_key	day_of_week	full_date_description	first_value
2003	1	Wednesday	January 1, 2003	January 1, 2003
2003	2	Thursday	January 2, 2003	January 1, 2003
2003	3	Friday	January 3, 2003	January 1, 2003
2003	4	Saturday	January 4, 2003	January 1, 2003
2003	5	Sunday	January 5, 2003	January 1, 2003
2003	6	Monday	January 6, 2003	January 1, 2003
2003	7	Tuesday	January 7, 2003	January 1, 2003
2003	8	Wednesday	January 8, 2003	January 1, 2003
2003	9	Thursday	January 9, 2003	January 1, 2003
2003	10	Friday	January 10, 2003	January 1, 2003
2003	11	Saturday	January 11, 2003	January 1, 2003
2003	12	Sunday	January 12, 2003	January 1, 2003
2003	13	Monday	January 13, 2003	January 1, 2003
2003	14	Tuesday	January 14, 2003	January 1, 2003
2003	15	Wednesday	January 15, 2003	January 1, 2003
2003	16	Thursday	January 16, 2003	January 1, 2003
2003	17	Friday	January 17, 2003	January 1, 2003
2003	18	Saturday	January 18, 2003	January 1, 2003
2003	19	Sunday	January 19, 2003	January 1, 2003
2003	20	Monday	January 20, 2003	January 1, 2003
2003	21	Tuesday	January 21, 2003	January 1, 2003
2003	22	Wednesday	January 22, 2003	January 1, 2003
2003	23	Thursday	January 23, 2003	January 1, 2003
2003	24	Friday	January 24, 2003	January 1, 2003
2003	25	Saturday	January 25, 2003	January 1, 2003
2003	26	Sunday	January 26, 2003	January 1, 2003
2003	27	Monday	January 27, 2003	January 1, 2003
2003	28	Tuesday	January 28, 2003	January 1, 2003
2003	29	Wednesday	January 29, 2003	January 1, 2003
2003	30	Thursday	January 30, 2003	January 1, 2003
2003	31	Friday	January 31, 2003	January 1, 2003
2003	32	Saturday	February 1, 2003	February 1, 2003
2003	33	Sunday	February 2, 2003	February 1, 2003
...				

(365 rows)

See Also

- [LAST_VALUE \[Analytic\]](#)
- [TIME_SLICE](#)
- [SQL Analytics](#)

LAG [Analytic]

Returns the value of the input expression at the given offset before the current row within a window. This function lets you access more than one row in a table at the same time. This is useful for comparing values when the relative positions of rows can be reliably known. It also lets you avoid the more costly self join, which enhances query processing speed.

For information on getting the rows that follow, see [LEAD](#).

Behavior Type

Immutable

Syntax

```
LAG ( expression [, offset ] [, default ] ) OVER (
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

<i>expression</i>	The expression to evaluate—for example, a constant, column, non-analytic function, function expression, or expressions involving any of these.
<i>offset</i>	Indicates how great is the lag. The default value is 1 (the previous row). This parameter must evaluate to a constant positive integer.
<i>default</i>	The value returned if <i>offset</i> falls outside the bounds of the table or partition. This value must be a constant value or an expression that can be evaluated to a constant; its data type is coercible to that of the first

	argument.
OVER()	See Analytic Functions

Examples

This example sums the current balance by date in a table and also sums the previous balance from the last day. Given the inputs that follow, the data satisfies the following conditions:

- For each `some_id`, there is exactly 1 row for each date represented by `month_date`.
- For each `some_id`, the set of dates is consecutive; that is, if there is a row for February 24 and a row for February 26, there would also be a row for February 25.
- Each `some_id` has the same set of dates.

```
=> CREATE TABLE balances (
    month_date DATE,
    current_bal INT,
    some_id INT);

=> INSERT INTO balances values ('2009-02-24', 10, 1);
=> INSERT INTO balances values ('2009-02-25', 10, 1);
=> INSERT INTO balances values ('2009-02-26', 10, 1);
=> INSERT INTO balances values ('2009-02-24', 20, 2);
=> INSERT INTO balances values ('2009-02-25', 20, 2);
=> INSERT INTO balances values ('2009-02-26', 20, 2);
=> INSERT INTO balances values ('2009-02-24', 30, 3);
=> INSERT INTO balances values ('2009-02-25', 20, 3);
=> INSERT INTO balances values ('2009-02-26', 30, 3);
```

Now run the `LAG()` function to sum the current balance for each date and sum the previous balance from the last day:

```
=> SELECT month_date,
    SUM(current_bal) as current_bal_sum,
    SUM(previous_bal) as previous_bal_sum FROM
    (SELECT month_date, current_bal,
    LAG(current_bal, 1, 0) OVER
    (PARTITION BY some_id ORDER BY month_date)
    AS previous_bal FROM balances) AS subQ
    GROUP BY month_date ORDER BY month_date;
month_date | current_bal_sum | previous_bal_sum
-----+-----+-----
2009-02-24 |          60 |           0
2009-02-25 |          50 |          60
2009-02-26 |          60 |          50
(3 rows)
```

Using the same example data, the following query would not be allowed because `LAG()` is nested inside an aggregate function:

```
=> SELECT month_date,
       SUM(current_bal) as current_bal_sum,
       SUM(LAG(current_bal, 1, 0) OVER
          (PARTITION BY some_id ORDER BY month_date)) AS previous_bal_sum
FROM some_table GROUP BY month_date ORDER BY month_date;
```

The following example uses the [VMart database](#). LAG first returns the annual income from the previous row, and then it calculates the difference between the income in the current row from the income in the previous row:

```
=> SELECT occupation, customer_key, customer_name, annual_income,
       LAG(annual_income, 1, 0) OVER (PARTITION BY occupation
          ORDER BY annual_income) AS prev_income, annual_income -
       LAG(annual_income, 1, 0) OVER (PARTITION BY occupation
          ORDER BY annual_income) AS difference
FROM customer_dimension ORDER BY occupation, customer_key LIMIT 20;
```

occupation	customer_key	customer_name	annual_income	prev_income	difference
Accountant	15	Midori V. Peterson	692610	692535	75
Accountant	43	Midori S. Rodriguez	282359	280976	1383
Accountant	93	Robert P. Campbell	471722	471355	367
Accountant	102	Sam T. McNulty	901636	901561	75
Accountant	134	Martha B. Overstreet	705146	704335	811
Accountant	165	James C. Kramer	376841	376474	367
Accountant	225	Ben W. Farmer	70574	70449	125
Accountant	270	Jessica S. Lang	684204	682274	1930
Accountant	273	Mark X. Lampert	723294	722737	557
Accountant	295	Sharon K. Gauthier	29033	28412	621
Accountant	338	Anna S. Jackson	816858	815557	1301
Accountant	377	William I. Jones	915149	914872	277
Accountant	438	Joanna A. McCabe	147396	144482	2914
Accountant	452	Kim P. Brown	126023	124797	1226
Accountant	467	Meghan K. Carcetti	810528	810284	244
Accountant	478	Tanya E. Greenwood	639649	639029	620
Accountant	511	Midori P. Vogel	187246	185539	1707
Accountant	525	Alexander K. Moore	677433	677050	383
Accountant	550	Sam P. Reyes	735691	735355	336
Accountant	577	Robert U. Vu	616101	615439	662

(20 rows)

The next example uses both LEAD and LAG to return the third row after the salary in the current row and fifth salary before the salary in the current row:

```
=> SELECT hire_date, employee_key, employee_last_name,
       LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "next_hired" ,
       LAG(hire_date, 1) OVER (ORDER BY hire_date) AS "last_hired"
FROM employee_dimension ORDER BY hire_date, employee_key;
```

hire_date	employee_key	employee_last_name	next_hired	last_hired
1956-04-11	2694	Farmer	1956-05-12	
1956-05-12	5486	Winkler	1956-09-18	1956-04-11
1956-09-18	5525	McCabe	1957-01-15	1956-05-12
1957-01-15	560	Greenwood	1957-02-06	1956-09-18
1957-02-06	9781	Bauer	1957-05-25	1957-01-15
1957-05-25	9506	Webber	1957-07-04	1957-02-06
1957-07-04	6723	Kramer	1957-07-07	1957-05-25

```
1957-07-07 |      5827 | Garnett      | 1957-11-11 | 1957-07-04
1957-11-11 |      373  | Reyes       | 1957-11-21 | 1957-07-07
1957-11-21 |     3874 | Martin      | 1958-02-06 | 1957-11-11
(10 rows)
```

See Also

- [LEAD \[Analytic\]](#)
- [SQL Analytics](#)

LAST_VALUE [Analytic]

Lets you select the last value of a table or partition (determined by the *window-order-clause*) without having to use a self join. LAST_VALUE takes the last record from the partition after the window order clause. The function then computes the expression against the last record, and returns the results. This function is useful when you want to use the last value as a baseline in calculations.

Use LAST_VALUE () with the *window-order-clause* to produce deterministic results. If no window is specified for the current row, the default window is UNBOUNDED PRECEDING AND CURRENT ROW.

Tip: Due to default window semantics, LAST_VALUE does not always return the last value of a partition. If you omit *Window Frame Clause* from the analytic clause, LAST_VALUE operates on this default window. Although results can seem non-intuitive by not returning the bottom of the current partition, it returns the bottom of the window, which continues to change along with the current input row being processed. If you want to return the last value of a partition, use UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING. See examples below.

Behavior Type

Immutable

Syntax

```
LAST_VALUE ( expression [ IGNORE NULLS ] ) OVER (
... [ window-partition-clause ]
```

... [[window-order-clause](#)]
... [[window-frame-clause](#)])

Parameters

<i>expression</i>	Expression to evaluate—for example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.
IGNORE NULLS	Specifies to return the last non-null value in the set, or NULL if all values are NULL. If you omit this option and the last value in the set is null, the function returns NULL.
OVER()	See Analytic Functions .

Example

Using the schema defined in [Window Framing](#) in Analyzing Data, the following query does not show the highest salary value by department; instead it shows the highest salary value by department by salary.

```
=> SELECT deptno, sal, empno, LAST_VALUE(sal)
      OVER (PARTITION BY deptno ORDER BY sal) AS lv
FROM emp;
deptno | sal | empno | lv
-----+-----+-----+-----
    10 | 101 |     1 | 101
    10 | 104 |     4 | 104
    20 | 100 |    11 | 100
    20 | 109 |     7 | 109
    20 | 109 |     6 | 109
    20 | 109 |     8 | 109
    20 | 110 |    10 | 110
    20 | 110 |     9 | 110
    30 | 102 |     2 | 102
    30 | 103 |     3 | 103
    30 | 105 |     5 | 105
```

If you include the window frame clause `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`, `LAST_VALUE()` returns the highest salary by department, an accurate representation of the information:

```
=> SELECT deptno, sal, empno, LAST_VALUE(sal)
      OVER (PARTITION BY deptno ORDER BY sal
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM emp;
deptno | sal | empno | lv
-----+-----+-----+-----
```

10		101		1		104
10		104		4		104
20		100		11		110
20		109		7		110
20		109		6		110
20		109		8		110
20		110		10		110
20		110		9		110
30		102		2		105
30		103		3		105
30		105		5		105

For more examples, see [FIRST_VALUE\(\)](#).

See Also

- [FIRST_VALUE \[Analytic\]](#)
- [TIME_SLICE](#)
- [SQL Analytics](#)

LEAD [Analytic]

Returns values from the row after the current row within a window, letting you access more than one row in a table at the same time. This is useful for comparing values when the relative positions of rows can be reliably known. It also lets you avoid the more costly self join, which enhances query processing speed.

Behavior Type

Immutable

Syntax

```
LEAD ( expression [, offset ] [, default ] ) OVER (
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

<i>expression</i>	The expression to evaluate—for example, a constant, column, non-analytic function, function expression, or expressions involving any of these.
<i>offset</i>	Is an optional parameter that defaults to 1 (the next row). This parameter must evaluate to a constant positive integer.
<i>default</i>	The value returned if <i>offset</i> falls outside the bounds of the table or partition. This value must be a constant value or an expression that can be evaluated to a constant; its data type is coercible to that of the first argument.
OVER()	See Analytic Functions

Examples

LEAD finds the hire date of the employee hired just after the current row:

```
=> SELECT employee_region, hire_date, employee_key, employee_last_name,
       LEAD(hire_date, 1) OVER (PARTITION BY employee_region ORDER BY hire_date) AS "next_hired"
       FROM employee_dimension ORDER BY employee_region, hire_date, employee_key;
employee_region | hire_date | employee_key | employee_last_name | next_hired
-----+-----+-----+-----+-----
East            | 1956-04-08 | 9218 | Harris            | 1957-02-06
East            | 1957-02-06 | 7799 | Stein             | 1957-05-25
East            | 1957-05-25 | 3687 | Farmer            | 1957-06-26
East            | 1957-06-26 | 9474 | Bauer             | 1957-08-18
East            | 1957-08-18 | 570  | Jefferson         | 1957-08-24
East            | 1957-08-24 | 4363 | Wilson            | 1958-02-17
East            | 1958-02-17 | 6457 | McCabe            | 1958-06-26
East            | 1958-06-26 | 6196 | Li                 | 1958-07-16
East            | 1958-07-16 | 7749 | Harris            | 1958-09-18
East            | 1958-09-18 | 9678 | Sanchez           | 1958-11-10
(10 rows)
```

The next example uses LEAD and LAG to return the third row after the salary in the current row and fifth salary before the salary in the current row.

```
=> SELECT hire_date, employee_key, employee_last_name,
       LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "next_hired" ,
       LAG(hire_date, 1) OVER (ORDER BY hire_date) AS "last_hired"
       FROM employee_dimension ORDER BY hire_date, employee_key;
hire_date | employee_key | employee_last_name | next_hired | last_hired
-----+-----+-----+-----+-----
1956-04-11 | 2694 | Farmer            | 1956-05-12 |
1956-05-12 | 5486 | Winkler           | 1956-09-18 | 1956-04-11
```

```

1956-09-18 |      5525 | McCabe           | 1957-01-15 | 1956-05-12
1957-01-15 |       560 | Greenwood       | 1957-02-06 | 1956-09-18
1957-02-06 |      9781 | Bauer           | 1957-05-25 | 1957-01-15
1957-05-25 |      9506 | Webber          | 1957-07-04 | 1957-02-06
1957-07-04 |      6723 | Kramer          | 1957-07-07 | 1957-05-25
1957-07-07 |      5827 | Garnett         | 1957-11-11 | 1957-07-04
1957-11-11 |       373 | Reyes           | 1957-11-21 | 1957-07-07
1957-11-21 |      3874 | Martin          | 1958-02-06 | 1957-11-11
(10 rows)

```

The following example returns employee name and salary, along with the next highest and lowest salaries.

```

=> SELECT employee_last_name, annual_salary,
       NVL(LEAD(annual_salary) OVER (ORDER BY annual_salary),
          MIN(annual_salary) OVER()) "Next Highest",
       NVL(LAG(annual_salary) OVER (ORDER BY annual_salary),
          MAX(annual_salary) OVER()) "Next Lowest"
  FROM employee_dimension;
employee_last_name | annual_salary | Next Highest | Next Lowest
-----+-----+-----+-----
Nielson            |          1200 |          1200 |      995533
Lewis              |          1200 |          1200 |          1200
Harris             |          1200 |          1202 |          1200
Robinson           |          1202 |          1202 |          1200
Garnett            |          1202 |          1202 |          1202
Weaver             |          1202 |          1202 |          1202
Nielson            |          1202 |          1202 |          1202
McNulty            |          1202 |          1204 |          1202
Farmer             |          1204 |          1204 |          1202
Martin             |          1204 |          1204 |          1204
(10 rows)

```

The next example returns, for each assistant director in the employees table, the hire date of the director hired just after the director on the current row. For example, Jackson was hired on 2016-12-28, and the next director hired was Bauer:

```

=> SELECT employee_last_name, hire_date,
       LEAD(hire_date, 1) OVER (ORDER BY hire_date DESC) as "NextHired"
  FROM employee_dimension WHERE job_title = 'Assistant Director';
employee_last_name | hire_date | NextHired
-----+-----+-----
Jackson            | 2016-12-28 | 2016-12-26
Bauer              | 2016-12-26 | 2016-12-11
Miller             | 2016-12-11 | 2016-12-07
Fortin             | 2016-12-07 | 2016-11-27
Harris             | 2016-11-27 | 2016-11-15
Goldberg           | 2016-11-15 |
(5 rows)

```

See Also

- [LAG \[Analytic\]](#)
- [SQL Analytics](#)

MAX [Analytic]

Returns the maximum value of an expression within a window. The return value has the same type as the expression data type.

The analytic functions `MIN()` and `MAX()` can operate with Boolean values. The `MAX()` function acts upon a [Boolean Data Type](#) or a value that can be implicitly converted to a Boolean value. If at least one input value is true, `MAX()` returns `t` (true). Otherwise, it returns `f` (false). In the same scenario, the `MIN()` function returns `t` (true) if all input values are true. Otherwise, it returns `f`.

Behavior Type

Immutable

Syntax

```
MAX ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any expression for which the maximum value is calculated, typically a column reference .
OVER()	See Analytic Functions .

Examples

The following query computes the deviation between the employees' annual salary and the maximum annual salary in Massachusetts:

```
=> SELECT employee_state, annual_salary,
       MAX(annual_salary)
       OVER(PARTITION BY employee_state ORDER BY employee_key) max,
       annual_salary- MAX(annual_salary)
       OVER(PARTITION BY employee_state ORDER BY employee_key) diff
FROM employee_dimension
WHERE employee_state = 'MA';
```

employee_state	annual_salary	max	diff
MA	1918	995533	-993615
MA	2058	995533	-993475
MA	2586	995533	-992947
MA	2500	995533	-993033
MA	1318	995533	-994215
MA	2072	995533	-993461
MA	2656	995533	-992877
MA	2148	995533	-993385
MA	2366	995533	-993167
MA	2664	995533	-992869

(10 rows)

The following example shows you the difference between the MIN and MAX analytic functions when you use them with a Boolean value. The sample creates a table with two columns, adds two rows of data, and shows sample output for MIN and MAX.

```
CREATE TABLE min_max_functions (emp VARCHAR, torf BOOL);

INSERT INTO min_max_functions VALUES ('emp1', 1);
INSERT INTO min_max_functions VALUES ('emp1', 0);

SELECT DISTINCT emp,
min(torf) OVER (PARTITION BY emp) AS worksasbooleanand,
Max(torf) OVER (PARTITION BY emp) AS worksasbooleanor
FROM min_max_functions;
```

emp	worksasbooleanand	worksasbooleanor
emp1	f	t

(1 row)

See Also

- [SQL Analytics](#)
- [MAX \[Aggregate\]](#)
- [MIN \[Analytic\]](#)

MEDIAN [Analytic]

For each row, returns the median value of a value set within each partition. MEDIAN determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data type, and returns that data type.

MEDIAN is an alias of [PERCENTILE_CONT \[Analytic\]](#) with an argument of 0.5 (50%).

Behavior Type

Immutable

Syntax

```
MEDIAN ( expression ) OVER ( [ window-partition-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the middle value or an interpolated value that would be the middle value once the values are sorted. Null values are ignored in the calculation.
OVER()	If the OVER clause specifies <i>window-partition-clause</i> , MEDIAN groups input rows according to one or more columns or expressions. If this clause is omitted, no grouping occurs and MEDIAN processes all input rows as a single partition.

Examples

See [Calculating a Median Value](#)

See Also

- [PERCENTILE_CONT \[Analytic\]](#)
- [SQL Analytics](#)

MIN [Analytic]

Returns the minimum value of an expression within a window. The return value has the same type as the expression data type.

The analytic functions `MIN()` and `MAX()` can operate with Boolean values. The `MAX()` function acts upon a [Boolean Data Type](#) or a value that can be implicitly converted to a Boolean value. If at least one input value is true, `MAX()` returns t (true). Otherwise, it returns f (false). In the same scenario, the `MIN()` function returns t (true) if all input values are true. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

```
MAX ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any expression for which the minimum value is calculated, typically a column reference .
OVER()	See Analytic Functions .

Examples

The following example shows how you can query to determine the deviation between the employees' annual salary and the minimum annual salary in Massachusetts:

```
=> SELECT employee_state, annual_salary,
        MIN(annual_salary)
        OVER(PARTITION BY employee_state ORDER BY employee_key) min,
        annual_salary - MIN(annual_salary)
        OVER(PARTITION BY employee_state ORDER BY employee_key) diff
FROM employee_dimension
WHERE employee_state = 'MA';
```

employee_state	annual_salary	min	diff
MA	1918	1204	714
MA	2058	1204	854
MA	2586	1204	1382
MA	2500	1204	1296
MA	1318	1204	114
MA	2072	1204	868
MA	2656	1204	1452
MA	2148	1204	944
MA	2366	1204	1162
MA	2664	1204	1460

(10 rows)

The following example shows you the difference between the MIN and MAX analytic functions when you use them with a Boolean value. The sample creates a table with two columns, adds two rows of data, and shows sample output for MIN and MAX.

```
CREATE TABLE min_max_functions (emp VARCHAR, torf BOOL);

INSERT INTO min_max_functions VALUES ('emp1', 1);
INSERT INTO min_max_functions VALUES ('emp1', 0);

SELECT DISTINCT emp,
min(torf) OVER (PARTITION BY emp) AS worksasbooleanand,
Max(torf) OVER (PARTITION BY emp) AS worksasbooleanor
FROM min_max_functions;
```

emp	worksasbooleanand	worksasbooleanor
emp1	f	t

(1 row)

See Also

- [SQL Analytics](#)
- [MIN \[Aggregate\]](#)
- [MAX \[Analytic\]](#)

NTILE [Analytic]

Equally divides an ordered data set (partition) into a *{value}* number of subsets within a window, where the subsets are numbered 1 through the value in parameter *constant-value*. For example, if *constant-value*= 4 and the partition contains 20 rows, NTILE divides the partition rows into four equal subsets of five rows. NTILE assigns each row to a subset by giving row a number from 1 to 4. The rows in the first subset are assigned 1, the next five are assigned 2, and so on.

If the number of partition rows is not evenly divisible by the number of subsets, the rows are distributed so no subset is more than one row larger than any other subset, and the lowest subsets have extra rows. For example, if *constant-value*= 4 and the number of rows = 21, the first subset has six rows, the second subset has five rows, and so on.

If the number of subsets is greater than the number of rows, then a number of subsets equal to the number of rows is filled, and the remaining subsets are empty.

Behavior Type

Immutable

Syntax

```
NTILE ( constant-value ) OVER (
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

<i>constant-value</i>	Specifies the number of subsets , where <i>constant-value</i> must
-----------------------	--

	resolve to a positive constant for each partition.
OVER()	See Analytic Functions .

Examples

The following query assigns each month's sales total into one of four subsets:

```
=> SELECT calendar_month_name AS MONTH, SUM(sales_quantity),
        NTILE(4) OVER (ORDER BY SUM(sales_quantity)) AS NTILE
FROM store.store_sales_fact JOIN date_dimension
USING(date_key)
GROUP BY calendar_month_name
ORDER BY NTILE;
MONTH | SUM | NTILE
-----+-----+-----
November | 2040726 | 1
June | 2088528 | 1
February | 2134708 | 1
April | 2181767 | 2
January | 2229220 | 2
October | 2316363 | 2
September | 2323914 | 3
March | 2354409 | 3
August | 2387017 | 3
July | 2417239 | 4
May | 2492182 | 4
December | 2531842 | 4
(12 rows)
```

See Also

- [PERCENTILE_CONT \[Analytic\]](#)
- [WIDTH_BUCKET](#)
- [SQL Analytics](#)

NTH_VALUE [Analytic]

Returns the value evaluated at the row that is the *n*th row of the window (counting from 1). If the specified row does not exist, NTH_VALUE returns NULL.

Behavior Type

Immutable

Syntax

```
NTH_VALUE ( expression, row-number [ IGNORE NULLS ] ) OVER (
... [ window-frame-clause ]
... [ Window Order Cause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Expression to evaluate. The expression can be a constant, column name, nonanalytic function, function expression, or expressions that include any of these.
<i>row-number</i>	Specifies the row to evaluate, where <i>row-number</i> evaluates to an integer ≥ 1 .
IGNORE NULLS	Specifies to return the first non-NULL value in the set, or NULL if all values are NULL.
OVER()	See Analytic Functions .

Example

In the following example, for each tuple (current row) in table t1, the window frame clause defines the window as follows:

```
ORDER BY b ROWS BETWEEN 3 PRECEDING AND CURRENT ROW
```

For each window, n for n th value is $a+1$. a is the value of column a in the tuple.

NTH_VALUE returns the result of the expression $b+1$, where b is the value of column b in the n th row, which is the $a+1$ row within the window.

```
=> SELECT * FROM t1 ORDER BY a;
 a | b
---+---
 1 | 10
 2 | 20
```

```
2 | 21
3 | 30
4 | 40
5 | 50
6 | 60
(7 rows)

=> SELECT NTH_VALUE(b+1, a+1) OVER
      (ORDER BY b ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) FROM t1;
?column?
-----
      22
      31

(7 rows)
```

PERCENT_RANK [Analytic]

Calculates the relative rank of a row for a given row in a group within a window by dividing that row's rank less 1 by the number of rows in the partition, also less 1. `PERCENT_RANK` always returns values from 0 to 1 inclusive. The first row in any set has a `PERCENT_RANK` of 0. The return value is `NUMBER`.

$$(\text{rank} - 1) / ([\text{rows}] - 1)$$

In the preceding formula, `rank` is the rank position of a row in the group and `rows` is the total number of rows in the partition defined by the `OVER()` clause.

Behavior Type

Immutable

Syntax

```
PERCENT_RANK ( ) OVER (
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

OVER()	See Analytic Functions
--------	--

Examples

The following example finds the percent rank of gross profit for different states within each month of the first quarter:

```
=> SELECT calendar_month_name AS MONTH, store_state,
        SUM(gross_profit_dollar_amount),
        PERCENT_RANK() OVER (PARTITION BY calendar_month_name
        ORDER BY SUM(gross_profit_dollar_amount)) AS PERCENT_RANK
FROM store.store_sales_fact JOIN date_dimension
USING(date_key)
JOIN store.store_dimension
USING (store_key)
WHERE calendar_month_name IN ('January','February','March')
AND store_state IN ('OR','IA','DC','NV','WI')
GROUP BY calendar_month_name, store_state
ORDER BY calendar_month_name, PERCENT_RANK;
```

MONTH	store_state	SUM	PERCENT_RANK
February	IA	418490	0
February	OR	460588	0.25
February	DC	616553	0.5
February	WI	619204	0.75
February	NV	838039	1
January	OR	446528	0
January	IA	474501	0.25
January	DC	628496	0.5
January	WI	679382	0.75
January	NV	871824	1
March	IA	460282	0
March	OR	481935	0.25
March	DC	716063	0.5
March	WI	771575	0.75
March	NV	970878	1

(15 rows)

The following example calculates, for each employee, the percent rank of the employee's salary by their job title:

```
=> SELECT job_title, employee_last_name, annual_salary,
        PERCENT_RANK()
        OVER (PARTITION BY job_title ORDER BY annual_salary DESC) AS percent_rank
FROM employee_dimension
ORDER BY percent_rank, annual_salary;
```

job_title	employee_last_name	annual_salary	percent_rank
-----------	--------------------	---------------	--------------

Cashier	Fortin	3196	0
Delivery Person	Garnett	3196	0
Cashier	Vogel	3196	0
Customer Service	Sanchez	3198	0
Shelf Stocker	Jones	3198	0
Custodian	Li	3198	0
Customer Service	Kramer	3198	0
Greeter	McNulty	3198	0
Greeter	Greenwood	3198	0
Shift Manager	Miller	99817	0
Advertising	Vu	99853	0
Branch Manager	Jackson	99858	0
Marketing	Taylor	99928	0
Assistant Director	King	99973	0
Sales	Kramer	99973	0
Head of PR	Goldberg	199067	0
Regional Manager	Gauthier	199744	0
Director of HR	Moore	199896	0
Head of Marketing	Overstreet	199955	0
VP of Advertising	Meyer	199975	0
VP of Sales	Sanchez	199992	0
Founder	Gauthier	927335	0
CEO	Taylor	953373	0
Investor	Garnett	963104	0
Co-Founder	Vu	977716	0
CFO	Vogel	983634	0
President	Sanchez	992363	0
Delivery Person	Li	3194	0.00114155251141553
Delivery Person	Robinson	3194	0.00114155251141553
Custodian	McCabe	3192	0.00126582278481013
Shelf Stocker	Moore	3196	0.00128040973111396
Branch Manager	Moore	99716	0.00186567164179104
...			

See Also

- [CUME_DIST \[Analytic\]](#)
- [SQL Analytics](#)

PERCENTILE_CONT [Analytic]

An inverse distribution function where, for each row, PERCENTILE_CONT returns the value that would fall into the specified percentile among a set of values in each partition within a window. For example, if the argument to the function is 0.5, the result of the function is the median of the data set (50th percentile). PERCENTILE_CONT assumes a continuous distribution data model. NULL values are ignored.

PERCENTILE_CONT computes the percentile by first computing the row number where the percentile row would exist. For example:

$$row-number = 1 + percentile-value * (num-partition-rows - 1)$$

If *row-number* is a whole number (within an error of 0.00001), the percentile is the value of row *row-number*.

Otherwise, Vertica interpolates the percentile value between the value of the CEILING (*row-number*) row and the value of the FLOOR(*row-number*) row. In other words, the percentile is calculated as follows:

$$\begin{aligned} & (\text{CEILING}(row-number) - row-number) * (\text{Value of FLOOR}(row-number) row) \\ & + (row-number - \text{FLOOR}(row-number)) * (\text{Value of CEILING}(row-number) row) \end{aligned}$$

Note: If the percentile value is 0.5, PERCENTILE_CONT returns the same result set as the function [MEDIAN](#).

Behavior Type

Immutable

Syntax

```
PERCENTILE_CONT ( percentile ) WITHIN GROUP ( ORDER BY expression [ ASC | DESC ] ) OVER (
... [ window-partition-clause ] )
```

Parameters

<i>percentile</i>	Percentile value, a FLOAT constant that ranges from 0 to 1 (inclusive).
WITHIN GROUP (ORDER BY <i>expression</i>)	Specifies how to sort data within each group. ORDER BY takes only one column/expression that must be INTEGER, FLOAT, INTERVAL, or NUMERIC data type. NULL values are discarded. The WITHIN GROUP (ORDER BY) clause does not guarantee the order of the SQL result. To order the final result, use the SQL ORDER BY clause set.
ASC DESC	Specifies the ordering sequence as ascending (default) or descending. Specifying ASC or DESC in the WITHIN GROUP clause affects results as long as the <i>percentile</i> is not 0.5.
OVER()	See Analytic Functions

Examples

This query computes the median annual income per group for the first 300 customers in Wisconsin and the District of Columbia.

```
=> SELECT customer_state, customer_key, annual_income,  
        PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY annual_income)  
        OVER (PARTITION BY customer_state) AS PERCENTILE_CONT  
FROM customer_dimension  
WHERE customer_state IN ('DC','WI')  
AND customer_key < 300  
ORDER BY customer_state, customer_key;  
customer_state | customer_key | annual_income | PERCENTILE_CONT  
-----+-----+-----+-----  
DC              |          52 |       168312 |          483266.5  
DC              |          118 |       798221 |          483266.5  
WI              |           62 |       283043 |          377691  
WI              |          139 |       472339 |          377691  
(4 rows)
```

See Also

- [MEDIAN \[Analytic\]](#)
- [SQL Analytics](#)

PERCENTILE_DISC [Analytic]

An inverse distribution function where, for each row, `PERCENTILE_DISC` returns the value that would fall into the specified percentile among a set of values in each partition within a window. `PERCENTILE_DISC()` assumes a discrete distribution data model. `NULL` values are ignored.

`PERCENTILE_DISC` examines the cumulative distribution values in each group until it finds one that is greater than or equal to the specified percentile. Vertica computes the percentile where, for each row, `PERCENTILE_DISC` outputs the first value of the `WITHIN GROUP (ORDER BY)` column whose `CUME_DIST` (cumulative distribution) value is \geq the argument `FLOAT` value—for example, `0.4`:

```
PERCENTILE_DIST(0.4) WITHIN GROUP (ORDER BY salary) OVER(PARTITION BY deptno)...
```

Given the following query:

```
SELECT CUME_DIST() OVER(ORDER BY salary) FROM table-name;
```

The smallest CUME_DIST value that is greater than 0.4 is also the PERCENTILE_DISC.

Behavior Type

Immutable

Syntax

```
PERCENTILE_DISC ( percentile ) WITHIN GROUP (
... ORDER BY expression [ ASC | DESC ] ) OVER (
... [ window-partition-clause ] )
```

Parameters

<i>percentile</i>	Percentile value, a FLOAT constant that ranges from 0 to 1 (inclusive).
WITHIN GROUP (ORDER BY <i>expression</i>)	Specifies how to sort data within each group. ORDER BY takes only one column/expression that must be INTEGER, FLOAT, INTERVAL, or NUMERIC data type. NULL values are discarded. The WITHIN GROUP (ORDER BY) clause does not guarantee the order of the SQL result. To order the final result, use the SQL ORDER BY clause set.
ASC DESC	Specifies the ordering sequence as ascending (default) or descending.
OVER()	See Analytic Functions

Example

This query computes the 20th percentile annual income by group for first 300 customers in Wisconsin and the District of Columbia.

```
=> SELECT customer_state, customer_key, annual_income,
        PERCENTILE_DISC(.2) WITHIN GROUP(ORDER BY annual_income)
        OVER (PARTITION BY customer_state) AS PERCENTILE_DISC
FROM customer_dimension
WHERE customer_state IN ('DC','WI')
AND customer_key < 300
ORDER BY customer_state, customer_key;
customer_state | customer_key | annual_income | PERCENTILE_DISC
-----+-----+-----+-----
DC              |          104 |      658383   |          417092
```

DC		168		417092		417092
DC		245		670205		417092
WI		106		227279		227279
WI		127		703889		227279
WI		209		458607		227279

(6 rows)

See Also

- [CUME_DIST \[Analytic\]](#)
- [PERCENTILE_CONT \[Analytic\]](#)
- [SQL Analytics](#)

RANK [Analytic]

Within each window partition, ranks all rows in the query results set according to the order specified by the window's `ORDER BY` clause.

RANK executes as follows:

1. Sorts partition rows as specified by the `ORDER BY` clause.
2. Compares the `ORDER BY` values of the preceding row and current row and ranks the current row as follows:
 - If `ORDER BY` values are the same, the current row gets the same ranking as the preceding row.

Note: Null values are considered equal. For detailed information on how null values are sorted, see [NULL Sort Order](#).
 - If the `ORDER BY` values are different, `DENSE_RANK` increments or decrements the current row's ranking by 1, plus the number of consecutive duplicate values in the rows that precede it.

The largest rank value is the equal to the total number of rows returned by the query.

Behavior Type

Immutable

Syntax

```
RANK() OVER(  
... [ window-partition-clause ]  
... window-order-clause )
```

Parameters

OVER()	See Analytic Functions
--------	--

Compared with DENSE_RANK

RANK can leave gaps in the ranking sequence, while DENSE_RANK does not. For more information, see [DENSE_RANK](#).

Examples

The following query ranks by state all company customers that have been customers since 2007. In rows where the `customer_since` dates are the same, RANK assigns the rows equal ranking. When the `customer_since` date changes, RANK skips one or more rankings—for example, within CA, from 12 to 14, and from 17 to 19.

```
=> SELECT customer_state, customer_name, customer_since,  
       RANK() OVER (PARTITION BY customer_state ORDER BY customer_since) AS rank  
   FROM customer_dimension WHERE customer_type='Company' AND customer_since > '01/01/2007'  
   ORDER BY customer_state;  
customer_state | customer_name | customer_since | rank  
-----+-----+-----+-----  
AZ              | Foodshop      | 2007-01-20    | 1  
AZ              | Goldstar      | 2007-08-11    | 2  
CA              | Metahope      | 2007-01-05    | 1  
CA              | Foodgen       | 2007-02-05    | 2  
CA              | Infohope      | 2007-02-09    | 3  
CA              | Foodcom       | 2007-02-19    | 4  
CA              | Amerihope     | 2007-02-22    | 5  
CA              | Infostar      | 2007-03-05    | 6  
CA              | Intracare     | 2007-03-14    | 7  
CA              | Infocare      | 2007-04-07    | 8  
...  
CO              | Goldtech      | 2007-02-19    | 1  
CT              | Foodmedia     | 2007-02-11    | 1  
CT              | Metatech      | 2007-02-20    | 2  
CT              | Infocorp      | 2007-04-10    | 3  
...
```

See Also

[SQL Analytics](#)

ROW_NUMBER [Analytic]

Assigns a sequence of unique numbers, starting from 1, to each row in a window partition. Use the optional window partition clause to group data into partitions before operating on it. For example:

```
SUM OVER (PARTITION BY col1, col2, ...)
```

Notes:

- ROW_NUMBER() is a Vertica extension, not part of the SQL-99 standard.
- ROW_NUMBER and RANK are generally interchangeable. ROW_NUMBER differs from RANK in that it assigns a unique ordinal number to each row in the ordered set, starting with 1.

Behavior Type

Immutable

Syntax

```
ROW_NUMBER ( ) OVER (  
... [ window-partition-clause ]  
... window-order-clause )
```

Parameters

OVER()	See Analytic Functions
--------	--

Examples

The following query partitions customers in the VMart table `customer_dimension` by occupation. It then ranks those customers according to the ordered set specified by the window partition clause.

```
=> SELECT occupation, customer_key, customer_since, annual_income,
       ROW_NUMBER() OVER (PARTITION BY occupation) AS customer_since_row_num
FROM public.customer_dimension
ORDER BY occupation, customer_since_row_num;
```

occupation	customer_key	customer_since	annual_income	customer_since_row_num
Accountant	49985	1987-04-05	998685	1
Accountant	49977	1989-06-14	616235	2
Accountant	49929	1978-10-08	298965	3
Accountant	49913	1988-10-18	141013	4
Accountant	49844	1989-10-04	967475	5
Accountant	49839	1971-11-08	942459	6
Accountant	49823	1979-11-28	435959	7
Accountant	49817	1987-06-15	538732	8
Accountant	49733	1972-04-21	651928	9
Accountant	49707	1980-02-17	420219	10
...				
Acrobat	49912	2007-06-08	722953	1
Acrobat	49908	1996-03-05	380288	2
Acrobat	49833	2003-11-15	317918	3
Acrobat	49770	1984-11-18	986536	4
Acrobat	49725	1973-04-09	911064	5
Acrobat	49641	1970-11-24	870287	6
Acrobat	49605	1972-01-08	322062	7
Acrobat	49577	1980-01-15	121727	8
Acrobat	49575	1975-11-03	835388	9
...				
Actor	49974	2003-06-12	885346	1
Actor	49937	1986-07-25	692557	2
Actor	49889	1985-09-09	766587	3
Actor	49850	1980-01-01	328270	4
Actor	49820	1984-11-17	826061	5
Actor	49780	1987-06-01	54853	6
Actor	49760	2000-07-05	255977	7
Actor	49698	1971-05-18	543584	8
Actor	49676	1997-07-23	710498	9
Actor	49631	1985-11-12	67353	10
...				

See Also

- [RANK \[Analytic\]](#)
- [SQL Analytics](#)

STDDEV [Analytic]

Computes the statistical sample standard deviation of the current row with respect to the group within a window. `STDDEV_SAMP` returns the same value as the square root of the variance defined for the `VAR_SAMP` function:

```
STDDEV( expression ) = SQRT(VAR_SAMP( expression ))
```

When `VAR_SAMP` returns NULL, this function returns NULL.

Note: The nonstandard function `STDDEV` is provided for compatibility with other databases. It is semantically identical to `STDDEV_SAMP`.

Behavior Type

Immutable

Syntax

```
STDDEV ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER()	See Analytic Functions

Example

The following example returns the standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,
        STDDEV(annual_salary) OVER (ORDER BY hire_date) as "stddev"
   FROM employee_dimension
  WHERE job_title = 'Assistant Director';
employee_last_name | annual_salary |      stddev
-----+-----+-----
Bauer              |      85003   |          NaN
Reyes              |      91051   | 4276.58181261624
Overstreet         |      53296   | 20278.6923394976
Gauthier           |      97216   | 19543.7184537642
Jones              |      82320   | 16928.0764028285
Fortin             |      56166   | 18400.2738421652
Carcetti           |      71135   | 16968.9453554483
Weaver             |      74419   | 15729.0709901852
Stein              |      85689   | 15040.5909495309
McNulty            |      69423   | 14401.1524291943
Webber             |      99091   | 15256.3160166536
Meyer              |      74774   | 14588.6126417355
Garnett            |      82169   | 14008.7223268494
Roy                |      76974   | 13466.1270356647
Dobisz             |      83486   | 13040.4887828347
Martin             |      99702   | 13637.6804131055
Martin             |      73589   | 13299.2838158566
...
```

See Also

- [STDDEV \[Aggregate\]](#)
- [STDDEV_SAMP \[Aggregate\]](#)
- [STDDEV_SAMP \[Analytic\]](#)
- [SQL Analytics](#)

STDDEV_POP [Analytic]

Computes the statistical population standard deviation and returns the square root of the population variance within a window. The `STDDEV_POP()` return value is the same as the square root of the `VAR_POP()` function:

```
STDDEV_POP( expression ) = SQRT(VAR_POP( expression ))
```

When `VAR_POP` returns null, `STDDEV_POP` returns null.

Behavior Type

Immutable

Syntax

```
STDDEV_POP ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER()	See Analytic Functions .

Examples

The following example returns the population standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,
        STDDEV_POP(annual_salary) OVER (ORDER BY hire_date) as "stddev_pop"
   FROM employee_dimension WHERE job_title = 'Assistant Director';
employee_last_name | annual_salary | stddev_pop
-----+-----+-----
Goldberg           |          61859 |           0
Miller             |          79582 |          8861.5
Goldberg           |          74236 | 7422.74712548456
Campbell          |          66426 | 6850.22125098891
Moore              |          66630 | 6322.08223926257
Nguyen             |          53530 | 8356.55480080699
Harris             |          74115 | 8122.72288970008
Lang               |          59981 | 8053.54776538731
Farmer             |          60597 | 7858.70140687825
Nguyen             |          78941 | 8360.63150784682
```

See Also

- [STDDEV_POP \[Aggregate\]](#)
- [SQL Analytics](#)

STDDEV_SAMP [Analytic]

Computes the statistical sample standard deviation of the current row with respect to the group within a window. `STDDEV_SAMP`'s return value is the same as the square root of the variance defined for the `VAR_SAMP` function:

```
STDDEV( expression ) = SQRT(VAR_SAMP( expression ))
```

When `VAR_SAMP` returns NULL, `STDDEV_SAMP` returns NULL.

Note: `STDDEV_SAMP()` is semantically identical to the nonstandard function, `STDDEV()`.

Behavior Type

Immutable

Syntax

```
STDDEV_SAMP ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument..
OVER()	See Analytic Functions

Examples

The following example returns the sample standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,  
        STDDEV(annual_salary) OVER (ORDER BY hire_date) as "stddev_samp"  
FROM employee_dimension WHERE job_title = 'Assistant Director';  
employee_last_name | annual_salary | stddev_samp
```

employee_last_name	annual_salary	stddev_samp
Bauer	85003	NaN
Reyes	91051	4276.58181261624
Overstreet	53296	20278.6923394976
Gauthier	97216	19543.7184537642
Jones	82320	16928.0764028285
Fortin	56166	18400.2738421652
Carcetti	71135	16968.9453554483
Weaver	74419	15729.0709901852
Stein	85689	15040.5909495309
McNulty	69423	14401.1524291943
Webber	99091	15256.3160166536
Meyer	74774	14588.6126417355
Garnett	82169	14008.7223268494
Roy	76974	13466.1270356647
Dobisz	83486	13040.4887828347
...		

See Also

- [Analytic Functions](#)
- [STDDEV \[Analytic\]](#)
- [STDDEV \[Aggregate\]](#)
- [STDDEV_SAMP \[Aggregate\]](#)
- [SQL Analytics](#)

SUM [Analytic]

Computes the sum of an expression over a group of rows within a window. It returns a DOUBLE PRECISION value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

Behavior Type

Immutable

Syntax

```
SUM ( expression ) OVER (
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER()	See Analytic Functions

Overflow Handling

If you encounter data overflow when using SUM, use [SUM_FLOAT](#) which converts data to a floating point. By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

Examples

The following query returns the cumulative sum all of the returns made to stores in January:

```
=> SELECT calendar_month_name AS month, transaction_type, sales_quantity,
SUM(sales_quantity)
OVER (PARTITION BY calendar_month_name ORDER BY date_dimension.date_key) AS SUM
FROM store.store_sales_fact JOIN date_dimension
USING(date_key) WHERE calendar_month_name IN ('January')
AND transaction_type= 'return';
 month | transaction_type | sales_quantity | SUM
-----+-----+-----+-----
January | return           |                | 7 | 651
January | return           |                | 3 | 651
```

January		return		7		651
January		return		7		651
January		return		7		651
January		return		3		651
January		return		7		651
January		return		5		651
January		return		1		651
January		return		6		651
January		return		6		651
January		return		3		651
January		return		9		651
January		return		7		651
January		return		6		651
January		return		8		651
January		return		7		651
January		return		2		651
January		return		4		651
January		return		5		651
January		return		7		651
January		return		8		651
January		return		4		651
January		return		10		651
January		return		6		651
...						

See Also

- [SUM \[Aggregate\]](#)
- [Numeric Data Types](#)
- [SQL Analytics](#)

VAR_POP [Analytic]

Returns the statistical population variance of a non-null set of numbers (nulls are ignored) in a group within a window. Results are calculated by the sum of squares of the difference of *expression* from the mean of *expression*, divided by the number of rows remaining:

```
(SUM( expression * expression ) - SUM( expression ) * SUM( expression ) / COUNT( expression )) /  
COUNT( expression )
```

Behavior Type

Immutable

Syntax

```
VAR_POP ( expression ) OVER (  
... [ window-partition-clause ]  
... [ window-order-clause ]  
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument
OVER()	See Analytic Functions

Examples

The following example calculates the cumulative population in the store orders fact table of sales in January 2007:

```
=> SELECT date_ordered,  
        VAR_POP(SUM(total_order_cost))  
        OVER (ORDER BY date_ordered) "var_pop"  
FROM store.store_orders_fact s  
WHERE date_ordered BETWEEN '2007-01-01' AND '2007-01-31'  
GROUP BY s.date_ordered;  
date_ordered |      var_pop  
-----+-----  
2007-01-01 |          0  
2007-01-02 |      89870400  
2007-01-03 |     3470302472  
2007-01-04 |    4466755450.6875  
2007-01-05 |    3816904780.80078  
2007-01-06 |     25438212385.25  
2007-01-07 |    22168747513.1016  
2007-01-08 |    23445191012.7344  
2007-01-09 |    39292879603.1113  
2007-01-10 |    48080574326.9609  
(10 rows)
```

See Also

- [VAR_POP \[Aggregate\]](#)
- [SQL Analytics](#)

VAR_SAMP [Analytic]

Returns the sample variance of a non-NULL set of numbers (NULL values in the set are ignored) for each row of the group within a window. Results are calculated as follows:

```
(SUM( expression * expression ) - SUM( expression ) * SUM( expression ) / COUNT( expression ) )  
/ (COUNT( expression ) - 1 )
```

This function and [VARIANCE](#) differ in one way: given an input set of one element, VARIANCE returns 0 and VAR_SAMP returns NULL.

Behavior Type

Immutable

Syntax

```
VAR_SAMP ( expression ) OVER (  
... [ window-partition-clause ]  
... [ window-order-clause ]  
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument
OVER()	See Analytic Functions

Null Handling

- VAR_SAMP returns the sample variance of a set of numbers after it discards the NULL values in the set.
- If the function is applied to an empty set, then it returns NULL.

Examples

The following example calculates the sample variance in the store orders fact table of sales in December 2007:

```
=> SELECT date_ordered,
         VAR_SAMP(SUM(total_order_cost))
         OVER (ORDER BY date_ordered) "var_samp"
FROM store.store_orders_fact s
WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'
GROUP BY s.date_ordered;
```

date_ordered	var_samp
2007-12-01	NaN
2007-12-02	90642601088
2007-12-03	48030548449.3359
2007-12-04	32740062504.2461
2007-12-05	32100319112.6992
2007-12-06	26274166814.668
2007-12-07	23017490251.9062
2007-12-08	21099374085.1406
2007-12-09	27462205977.9453
2007-12-10	26288687564.1758

(10 rows)

See Also

- [VARIANCE \[Analytic\]](#)
- [VAR_SAMP \[Aggregate\]](#)
- [SQL Analytics](#)

VARIANCE [Analytic]

Returns the sample variance of a non-NULL set of numbers (NULL values in the set are ignored) for each row of the group within a window. Results are calculated as follows:

```
( SUM( expression * expression ) - SUM( expression ) * SUM( expression ) / COUNT( expression ) ) /  
(COUNT( expression ) - 1 )
```

VARIANCE returns the variance of *expression*, which is calculated as follows:

- 0 if the number of rows in *expression* = 1
- [VAR_SAMP](#) if the number of rows in *expression* > 1

Note: The nonstandard function VARIANCE is provided for compatibility with other databases. It is semantically identical to [VAR_SAMP](#).

Behavior Type

Immutable

Syntax

```
VAR_SAMP ( expression ) OVER (  
... [ window-partition-clause ]  
... [ window-order-clause ]  
... [ window-frame-clause ] )
```

Parameters

<i>expression</i>	Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.
OVER()	See Analytic Functions

Examples

The following example calculates the cumulative variance in the store orders fact table of sales in December 2007:

```
=> SELECT date_ordered,  
       VARIANCE(SUM(total_order_cost))  
       OVER (ORDER BY date_ordered) "variance"  
FROM store.store_orders_fact s
```

```
WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'  
GROUP BY s.date_ordered;  
date_ordered |      variance  
-----  
2007-12-01 |              NaN  
2007-12-02 |      2259129762  
2007-12-03 | 1809012182.33301  
2007-12-04 |   35138165568.25  
2007-12-05 | 26644110029.3003  
2007-12-06 |      25943125234  
2007-12-07 | 23178202223.9048  
2007-12-08 | 21940268901.1431  
2007-12-09 | 21487676799.6108  
2007-12-10 | 21521358853.4331  
(10 rows)
```

See Also

- [VAR_SAMP \[Analytic\]](#)
- [VARIANCE \[Aggregate\]](#)
- [VAR_SAMP \[Aggregate\]](#)
- [SQL Analytics](#)

Current Load Source

The current load source function returns the source file name.

CURRENT_LOAD_SOURCE

Returns the file name used during the COPY statement.

Behavior Type

Stable

Syntax

```
CURRENT_LOAD_SOURCE()
```

Behavior

- If the function is called outside of the context of a COPY statement, it returns NULL.
- If the function is called by a UDL that does not set the source, it returns the string <unknown>.

Examples

This example creates a table and populates column c3 with the names of the two separate files being loaded.

```
=> CREATE TABLE t (c1 integer, c2 varchar(50), c3 varchar(200));
CREATE TABLE

=> COPY t (c1, c2, c3 AS CURRENT_LOAD_SOURCE()) FROM '/home/load_file_1' ON exampledb_node02,
'/home/load_file_2' ON exampledb_node03 DELIMITER ',';

Rows Loaded
-----
5
(1 row)

=> SELECT * FROM t;
c1 | c2 | c3
-----+-----+-----
2 | dogs | load_file_1
1 | cats | load_file_1
4 | superheroes | load_file_2
3 | birds | load_file_1
5 | whales | load_file_2
(5 rows)
```

See Also

- [COPY](#)

Date/Time Functions

Date and time functions perform conversion, extraction, or manipulation operations on date and time data types and can return date and time information.

Usage

Functions that take `TIME` or `TIMESTAMP` inputs come in two variants:

- `TIME WITH TIME ZONE` or `TIMESTAMP WITH TIME ZONE`
- `TIME WITHOUT TIME ZONE` or `TIMESTAMP WITHOUT TIME ZONE`

For brevity, these variants are not shown separately.

The `+` and `*` operators come in commutative pairs; for example, both `DATE + INTEGER` and `INTEGER + DATE`. We show only one of each such pair.

Daylight Savings Time Considerations

When adding an `INTERVAL` value to (or subtracting an `INTERVAL` value from) a `TIMESTAMP WITH TIME ZONE` value, the days component advances (or decrements) the date of the `TIMESTAMP WITH TIME ZONE` by the indicated number of days. Across daylight saving time changes (with the session time zone set to a time zone that recognizes DST), this means `INTERVAL '1 day'` does not necessarily equal `INTERVAL '24 hours'`.

For example, with the session time zone set to `CST7CDT`:

```
TIMESTAMP WITH TIME ZONE '2014-04-02 12:00-07' + INTERVAL '1 day'
```

produces

```
TIMESTAMP WITH TIME ZONE '2014-04-03 12:00-06'
```

Adding `INTERVAL '24 hours'` to the same initial `TIMESTAMP WITH TIME ZONE` produces

```
TIMESTAMP WITH TIME ZONE '2014-04-03 13:00-06',
```

This result occurs because there is a change in daylight saving time at `2014-04-03 02:00` in time zone `CST7CDT`.

Date/Time Functions in Transactions

Certain date/time functions such as [CURRENT_TIMESTAMP](#) and [NOW](#) return the start time of the current transaction; for the duration of that transaction, they return the same value. Other date/time functions such as [TIMEOFDAY](#) always return the current time.

See Also

[Template Patterns for Date/Time Formatting](#)

ADD_MONTHS

Adds the specified number of months to a date and returns the sum as a DATE. The function returns the last day of the month if one of the following conditions is true:

- The *start-date* argument specifies the last day of the month. For example, `2015-01-28 +12` returns `2016-01-29` .
- The *start-date* argument's day component is greater than the last day of the month returned by `ADD_MONTHS`. For example, `2015-01-29 +1` returns `2015-02-28`.

Otherwise, `ADD_MONTHS` returns a date with the same day component as *start-date*. Thus, `2016-03-15 +2` returns `2016-05-15`.

Behavior Type

- Immutable if the *start-date* argument is a `TIMESTAMP` or `DATE`
- Stable if the *start-date* argument is a `TIMESTAMPTZ`

Syntax

```
ADD_MONTHS ( start-date, num-months );
```

Parameters

<i>start-date</i>	The date to process, an expression that evaluates to one of the following data types: <ul style="list-style-type: none">• DATE• TIMESTAMP• TIMESTAMPTZ
<i>num-months</i>	An integer expression that specifies the number of months to add to or subtract from <i>start-date</i> .

Examples

Add one month to the current date:

```
=> SELECT CURRENT_DATE Today;
      Today
-----
2016-05-05
(1 row)

VMart=> SELECT ADD_MONTHS(CURRENT_TIMESTAMP,1);
      ADD_MONTHS
-----
2016-06-05
(1 row)
```

Subtract four months from the current date:

```
=> SELECT ADD_MONTHS(CURRENT_TIMESTAMP, -4);
      ADD_MONTHS
-----
2016-01-05
(1 row)
```

Add one month to January 31 2016:

```
=> SELECT ADD_MONTHS('31-Jan-2016'::TIMESTAMP, 1) "Leap Month";
      Leap Month
-----
2016-02-29
(1 row)
```

The following example sets the timezone to EST; it then adds 24 months to a TIMESTAMPTZ that specifies a PST time zone, so ADD_MONTHS takes into account the time change:

```
=> SET TIME_ZONE 'America/New_York';
SET
VMart=> SELECT ADD_MONTHS('2008-02-29 23:30 PST'::TIMESTAMPTZ, 24);
  ADD_MONTHS
-----
 2010-03-01
(1 row)
```

AGE_IN_MONTHS

Returns the difference in months between two dates, expressed as an integer.

Behavior Type

- Immutable if both date arguments are of data type `TIMESTAMP`
- Stable if either date is a `TIMESTAMPTZ` or only one argument is supplied

Syntax

```
AGE_IN_MONTHS ( [ date1, ] date2 )
```

Parameters

<i>date1</i> <i>date2</i>	<p>Specify the boundaries of the period to measure. If you supply only one argument, Vertica sets <i>date1</i> to <code>CURRENT_DATE</code>. Both parameters must evaluate to one of the following data types:</p> <ul style="list-style-type: none">• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code> <p>If <i>date1</i> < <i>date2</i>, <code>AGES_IN_MONTHS</code> returns a negative value.</p>
------------------------------	--

Examples

Get the age in months of someone born March 2 1972, as of June 21 1990:

```
=> SELECT AGE_IN_MONTHS('1990-06-21'::TIMESTAMP, '1972-03-02'::TIMESTAMP);
   AGE_IN_MONTHS
-----
                219
(1 row)
```

Get the age in months of someone who was born November 21 1939, as of today:

```
=> SELECT AGE_IN_MONTHS ('1939-11-21'::DATE);
   AGE_IN_MONTHS
-----
                930
(1 row)
```

AGE_IN_YEARS

Returns the difference in years between two dates, expressed as an integer.

Behavior Type

- Immutable if both date arguments are of data type `TIMESTAMP`
- Stable if either date is a `TIMESTAMPTZ` or only one argument is supplied

Syntax

```
AGE_IN_YEARS( [ date1, ] date2 )
```

Parameters

<i>date1</i> <i>date2</i>	<p>Specify the boundaries of the period to measure. If you supply only one argument, Vertica sets <i>date1</i> to <code>CURRENT_DATE</code>. Both parameters must evaluate to one of the following data types:</p> <ul style="list-style-type: none">• <code>DATE</code>
------------------------------	--

- `TIMESTAMP`
- `TIMESTAMPTZ`

If `date1 < date2`, `AGES_IN_YEARS` returns a negative value.

Examples

Get the age of someone born March 2 1972, as of June 21 1990:

```
=> SELECT AGES_IN_YEARS('1990-06-21'::TIMESTAMP, '1972-03-02'::TIMESTAMP);
 AGES_IN_YEARS
-----
              18
(1 row)
```

Get the age of someone who was born November 21 1939, as of today:

```
=> SELECT AGES_IN_YEARS('1939-11-21'::DATE);
 AGES_IN_YEARS
-----
              77
(1 row)
```

CLOCK_TIMESTAMP

Returns a value of type `TIMESTAMP WITH TIMEZONE` that represents the current system-clock time.

`CLOCK_TIMESTAMP` uses the date and time supplied by the operating system on the server to which you are connected, which should be the same across all servers. The value changes each time you call it.

Behavior Type

Volatile

Syntax

```
CLOCK_TIMESTAMP()
```

Examples

The following command returns the current time on your system:

```
SELECT CLOCK_TIMESTAMP() "Current Time";
      Current Time
-----
2010-09-23 11:41:23.33772-04
(1 row)
```

Each time you call the function, you get a different result. The difference in this example is in microseconds:

```
SELECT CLOCK_TIMESTAMP() "Time 1", CLOCK_TIMESTAMP() "Time 2";
      Time 1          |          Time 2
-----+-----
2010-09-23 11:41:55.369201-04 | 2010-09-23 11:41:55.369202-04
(1 row)
```

See Also

- [STATEMENT_TIMESTAMP](#)
- [TRANSACTION_TIMESTAMP](#)

CURRENT_DATE

Returns the date (date-type value) on which the current transaction started.

Behavior Type

Stable

Syntax

```
CURRENT_DATE()
```

Note: You can call this function without parentheses.

Examples

```
SELECT CURRENT_DATE;  
?column?  
-----  
2010-09-23  
(1 row)
```

CURRENT_TIME

Returns a value of type `TIME WITH TIMEZONE` that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to `CURRENT_TIME` within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

```
CURRENT_TIME [ ( precision ) ]
```

Note: If you specify a column label without precision, you must also omit parentheses.

Parameters

<i>precision</i>	An integer value between 0-6, specifies to round the seconds fraction field result to the specified number of digits.
------------------	---

Examples

```
=> SELECT CURRENT_TIME(1) AS Time;  
Time  
-----  
06:51:45.2-07
```

```
(1 row)
=> SELECT CURRENT_TIME(5) AS Time;
      Time
-----
06:51:45.18435-07
(1 row)
```

CURRENT_TIMESTAMP

Returns a value of type `TIME WITH TIMEZONE` that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to `CURRENT_TIMESTAMP` within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

```
CURRENT_TIMESTAMP ( precision )
```

Parameters

<i>precision</i>	An integer value between 0-6, specifies to round the seconds fraction field result to the specified number of digits.
------------------	---

Examples

```
=> SELECT CURRENT_TIMESTAMP(1) AS time;
      time
-----
2017-03-27 06:50:49.7-07
(1 row)
=> SELECT CURRENT_TIMESTAMP(5) AS time;
      time
-----
2017-03-27 06:50:49.69967-07
(1 row)
```

DATE_PART

Extracts a sub-field such as year or hour from a date/time expression, equivalent to the the SQL-standard function [EXTRACT](#).

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `INTERVAL`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
DATE_PART ( 'field', date )
```

Parameters

<i>field</i>	A constant value that specifies the sub-field to extract from <i>date</i> (see Field Values below).
<i>date</i>	The date to process, an expression that evaluates to one of the following data types: <ul style="list-style-type: none">• <code>DATE</code> (cast to <code>TIMESTAMP</code>)• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>• <code>INTERVAL</code>

Field Values

CENTURY	The century number. The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number
---------	--

	0, you go from -1 to 1.
DAY	The day (of the month) field (1–31).
DECADE	The year field divided by 10.
DOQ	The day within the current quarter. DOQ recognizes leap year days.
DOW	Zero-based day of the week, where Sunday=0. Note: EXTRACT's day of week numbering differs from the function TO_CHAR .
DOY	The day of the year (1–365/366)
EPOCH	Specifies to return one of the following: <ul style="list-style-type: none"> • For DATE and TIMESTAMP values: the number of seconds before or since 1970-01-01 00:00:00-00 (if before, a negative number). • For INTERVAL values, the total number of seconds in the interval.
HOUR	The hour field (0–23).
ISODOW	The ISO day of the week , an integer between 1 and 7 where Monday is 1.
ISOWEEK	The ISO week of the year, an integer between 1 and 53.
ISOYEAR	The ISO year .
MICROSECONDS	The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds.
MILLENNIUM	The millennium number, where the first millennium is 1 and each millenium starts on 01-01-y001. For example, millennium 2 starts on 01-01-1001.
MILLISECONDS	The seconds field, including fractional parts, multiplied by 1000. This includes full seconds.
MINUTE	The minutes field (0 - 59).

MONTH	For <code>TIMESTAMP</code> values, the number of the month within the year (1 - 12) ; for <code>interval</code> values the number of months, modulo 12 (0 - 11).
QUARTER	The calendar quarter of the specified date as an integer, where the January-March quarter is 1, valid only for <code>TIMESTAMP</code> values.
SECOND	The seconds field, including fractional parts, 0–59, or 0-60 if the operating system implements leap seconds.
TIME_ZONE	The time zone offset from UTC, in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.
TIMEZONE_HOUR	The hour component of the time zone offset.
TIMEZONE_MINUTE	The minute component of the time zone offset.
WEEK	The number of the week of the calendar year that the day is in.
YEAR	The year field. There is no 0 AD, so subtract BC years from AD years accordingly.

Examples

Extract the day value:

```
SELECT DATE_PART('DAY', TIMESTAMP '2009-02-24 20:38:40') "Day";
   Day
-----
   24
(1 row)
```

Extract the month value:

```
SELECT DATE_PART('MONTH', '2009-02-24 20:38:40'::TIMESTAMP) "Month";
   Month
-----
     2
(1 row)
```

Extract the year value:

```
SELECT DATE_PART('YEAR', '2009-02-24 20:38:40'::TIMESTAMP) "Year";
   Year
-----
  2009
(1 row)
```

Extract the hours:

```
SELECT DATE_PART('HOUR', '2009-02-24 20:38:40'::TIMESTAMP) "Hour";
Hour
-----
    20
(1 row)
```

Extract the minutes:

```
SELECT DATE_PART('MINUTES', '2009-02-24 20:38:40'::TIMESTAMP) "Minutes";
Minutes
-----
     38
(1 row)
```

Extract the day of quarter (DOQ):

```
SELECT DATE_PART('DOQ', '2009-02-24 20:38:40'::TIMESTAMP) "DOQ";
DOQ
-----
   55
(1 row)
```

DATE

Converts the input value to a [DATE](#) data type.

Behavior Type

- Immutable if the input value is a `TIMESTAMP`, `DATE`, `VARCHAR`, or integer
- Stable if the input value is a `TIMESTAMPTZ`

Syntax

```
DATE ( value )
```

Parameters

<i>value</i>	The value to convert, one of the following: <ul style="list-style-type: none">• <code>TIMESTAMP</code>, <code>TIMESTAMPTZ</code>, <code>VARCHAR</code>, or another <code>DATE</code>.
--------------	---

- Integer: Vertica treats the integer as the number of days since 01/01/0001 and returns the date.

Examples

```
=> SELECT DATE (1);
      DATE
-----
0001-01-01
(1 row)

=> SELECT DATE (734260);
      DATE
-----
2011-05-03
(1 row)

=> SELECT DATE('TODAY');
      DATE
-----
2016-12-07
(1 row)
```

See Also

- [TO_DATE](#)
- [TO_TIMESTAMP](#)
- [TO_TIMESTAMP_TZ](#)

DATE_TRUNC

Truncates date and time values to the specified precision.

The return value is of type TIME or TIMETZ. All fields that are less than the specified precision are set to 0, or to 1 for day and month.

Behavior Type

Stable

Syntax

`DATE_TRUNC(precision, trunc-target)`

Parameters

<i>precision</i>	A string constant that specifies precision for the truncated value. See Precision Field Values below.
<i>trunc-target</i>	Value expression of type TIME or TIMETZ.

Precision Field Values

MILLENNIUM	The millennium number.
CENTURY	The century number. The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries.
DECADE	The year field divided by 10.
YEAR	The year field. Keep in mind there is no 0 AD, so subtract BC years from AD years with care.
MONTH	For <code>timestamp</code> values, the number of the month within the year (1–12) ; for <code>interval</code> values the number of months, modulo 12 (0–11).
WEEK	The number of the week of the year that the day is in. According to the ISO-8601 standard, the week starts on Monday, and the first week of a year contains January 4. Thus, an early January date can sometimes be in the week 52 or 53 of the previous calendar year. For example: <pre>=> SELECT YEAR_ISO('01-01-2016'::DATE), WEEK_ISO('01-01-2016'), DAYOFWEEK_ISO('01-01-2016'); YEAR_ISO WEEK_ISO DAYOFWEEK_ISO -----+-----+----- 2015 53 5 (1 row)</pre>

DAY	The day (of the month) field (1–31).
HOUR	The hour field (0–23).
MINUTE	The minutes field (0–59).
SECOND	The seconds field, including fractional parts (0–59) (60 if leap seconds are implemented by the operating system).
MILLISECOND S	The seconds field, including fractional parts, multiplied by 1000. Note that this includes full seconds.
MICROSECOND S	The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds.

Examples

The following example sets the field value as hour and returns the hour, truncating the minutes and seconds:

```
=> SELECT DATE_TRUNC('HOUR', TIMESTAMP '2012-02-24 13:38:40') AS HOUR;
      HOUR
-----
2012-02-24 13:00:00
(1 row)
```

The following example returns the year from the input `timestampz '2012-02-24 13:38:40'`. The function also defaults the month and day to January 1, truncates the hour:minute:second of the timestamp, and appends the time zone (`-05`):

```
=> SELECT DATE_TRUNC('YEAR', TIMESTAMPTZ '2012-02-24 13:38:40') AS YEAR;
      YEAR
-----
2012-01-01 00:00:00-05
(1 row)
```

The following example returns the year and month and defaults day of month to 1, truncating the rest of the string:

```
=> SELECT DATE_TRUNC('MONTH', TIMESTAMP '2012-02-24 13:38:40') AS MONTH;
      MONTH
-----
2012-02-01 00:00:00
(1 row)
```

DATEDIFF

Returns the time span between two dates, in the intervals specified. DATEDIFF excludes the start date in its calculation.

Behavior Type

- Immutable if start and end dates are TIMESTAMPT , DATE, TIME, or INTERVAL
- Stable if start and end dates are TIMESTAMPTZ

Syntax

```
DATEDIFF ( datepart, start, end );
```

Parameters

<i>datepart</i>	<p>Specifies the type of date or time intervals that DATEDIFF returns. If <i>datepart</i> is an expression, it must be enclosed in parentheses:</p> <pre>DATEDIFF(<i>expression</i>), <i>start</i>, <i>end</i>;</pre> <p><i>datepart</i> must evaluate to one of the following string literals, either quoted or unquoted:</p> <ul style="list-style-type: none">• year yy yyyy• quarter qq q• month mm m• day dayofyear dd d dy y• week wk ww• hour hh• minute mi n• second ss s
-----------------	--

	<ul style="list-style-type: none"> • millisecond ms • microsecond mcs us
<i>start, end</i>	<p>Specify the start and end dates, where <i>start</i> and <i>end</i> evaluate to one of the following data types:</p> <ul style="list-style-type: none"> • TIMESTAMP/TIMESTAMPZ • DATE • TIMESTAMPTZ • TIME/TIMETZ • INTERVAL <p>If $end < start$, DATEDIFF returns a negative value.</p> <p>Note: TIME and INTERVAL data types are invalid for start and end dates if <i>datepart</i> is set to year, quarter, or month.</p>

Compatible Start and End Date Data Types

The following table shows which data types can be matched as start and end dates:

	DATE	TIMESTAMP	TIMESTAMPZ	TIME	INTERVAL
DATE	•	•	•		
TIMESTAMP	•	•	•		
TIMESTAMPZ	•	•	•		
TIME				•	
INTERVAL					•

For example, if you set the start date to an INTERVAL data type, the end date must also be an INTERVAL, otherwise Vertica returns an error:

```
SELECT DATEDIFF(day, INTERVAL '26 days', INTERVAL '1 month ');
datediff
```

```
-----  
      4  
(1 row)
```

Date Part Intervals

DATEDIFF uses the *datepart* argument to calculate the number of intervals between two dates, rather than the actual amount of time between them. DATEDIFF uses the following cutoff points to calculate those intervals:

- year: January 1
- quarter: January 1, April 1, July 1, October 1
- month: the first day of the month
- week: Sunday at midnight (24:00)

For example, if *datepart* is set to year, DATEDIFF uses January 01 to calculate the number of years between two dates. The following DATEDIFF statement sets *datepart* to year, and specifies a time span 01/01/2005 - 06/15/2008:

```
SELECT DATEDIFF(year, '01-01-2005'::date, '12-31-2008'::date);  
datediff  
-----  
      3  
(1 row)
```

DATEDIFF always excludes the start date when it calculates intervals—in this case, 01/01//2005. DATEDIFF considers only calendar year starts in its calculation, so in this case it only counts years 2006, 2007, and 2008. The function returns 3, although the actual time span is nearly four years.

If you change the start and end dates to 12/31/2004 and 01/01/2009, respectively, DATEDIFF also counts years 2005 and 2009. This time, it returns 5, although the actual time span is just over four years:

```
=> SELECT DATEDIFF(year, '12-31-2004'::date, '01-01-2009'::date);  
datediff  
-----  
      5  
(1 row)
```

Similarly, DATEDIFF uses month start dates when it calculates the number of months between two dates. Thus, given the following statement, DATEDIFF counts months February through September and returns 8:

```
=> SELECT DATEDIFF(month, '01-31-2005'::date, '09-30-2005'::date);
datediff
-----
        8
(1 row)
```

See Also

[TIMESTAMPDIFF](#)

DAY

Returns as an integer the day of the month from the input value.

Behavior Type

- Immutable if the input value is a `TIMESTAMP`, `DATE`, `VARCHAR`, or `INTEGER`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
DAY ( value )
```

Parameters

<i>value</i>	The value to convert, one of the following: <code>TIMESTAMP</code> , <code>TIMESTAMPTZ</code> , <code>INTERVAL</code> , <code>VARCHAR</code> , or <code>INTEGER</code> .
--------------	--

Examples

```
=> SELECT DAY (6);
DAY
-----
    6
(1 row)

=> SELECT DAY(TIMESTAMP 'sep 22, 2011 12:34');
```

```
DAY
-----
 22
(1 row)

=> SELECT DAY('sep 22, 2011 12:34');
DAY
-----
 22
(1 row)

=> SELECT DAY(INTERVAL '35 12:34');
DAY
-----
 35
(1 row)
```

DAYOFMONTH

Returns the day of the month as an integer.

Behavior Type

- Immutable if the target date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the target date is a `TIMESTAMPTZ`

Syntax

```
DAYOFMONTH ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	--

Example

```
=> SELECT DAYOFMONTH (TIMESTAMP 'sep 22, 2011 12:34');
DAYOFMONTH
-----
                22
(1 row)
```

DAYOFWEEK

Returns the day of the week as an integer, where Sunday is day 1.

Behavior Type

- Immutable if the target date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the target date is `aTIMESTAMPTZ`

Syntax

```
DAYOFWEEK ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	--

Example

```
=> SELECT DAYOFWEEK (TIMESTAMP 'sep 17, 2011 12:34');
DAYOFWEEK
-----
              7
(1 row)
```

DAYOFWEEK_ISO

Returns the ISO 8061 day of the week as an integer, where Monday is day 1.

Behavior Type

- Immutable if the target date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the target date is `aTIMESTAMPTZ`

Syntax

```
DAYOFWEEK_ISO ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ
-------------	--

Examples

```
=> SELECT DAYOFWEEK_ISO(TIMESTAMP 'Sep 22, 2011 12:34');
DAYOFWEEK_ISO
-----
                4
(1 row)
```

The following example shows how to combine the DAYOFWEEK_ISO, WEEK_ISO, and YEAR_ISO functions to find the ISO day of the week, week, and year:

```
=> SELECT DAYOFWEEK_ISO('Jan 1, 2000'), WEEK_ISO('Jan 1, 2000'), YEAR_ISO('Jan1,2000');
DAYOFWEEK_ISO | WEEK_ISO | YEAR_ISO
-----+-----+-----
                6 |         52 |      1999
(1 row)
```

See Also

- [WEEK_ISO](#)
- [DAYOFWEEK_ISO](#)
- http://en.wikipedia.org/wiki/ISO_8601

DAYOFYEAR

Returns the day of the year as an integer, where January 1 is day 1.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPPTZ`

Syntax

```
DAYOFYEAR ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ
-------------	--

Example

```
=> SELECT DAYOFYEAR (TIMESTAMP 'SEPT 22,2011 12:34');
DAYOFYEAR
-----
          265
(1 row)
```

DAYS

Returns the integer value of the specified date, where 1 AD is 1. If the date precedes 1 AD, DAYS returns a negative integer.

Behavior Type

- Immutable if the specified date is a [TIMESTAMP](#), [DATE](#), or [VARCHAR](#)
- Stable if the specified date is a [TIMESTAMPTZ](#)

Syntax

```
DAYS ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ
-------------	--

Example

```
=> SELECT DAYS (DATE '2011-01-22');
   DAYS
-----
 734159
(1 row)

=> SELECT DAYS (DATE 'March 15, 0044 BC');
   DAYS
-----
-15997
(1 row)
```

EXTRACT

Retrieves sub-fields such as year or hour from date/time values and returns values of type [NUMERIC](#). EXTRACT is intended for computational processing, rather than for formatting date/time values for display.

Behavior Type

- Immutable if the specified date is a [TIMESTAMP](#), [DATE](#), or [INTERVAL](#)
- Stable if the specified date is a [TIMESTAMPTZ](#)

Syntax

EXTRACT (*field* FROM *date*)

Parameters

<i>field</i>	A constant value that specifies the sub-field to extract from <i>date</i> (see Field Values below).
<i>date</i>	The date to process, an expression that evaluates to one of the following data types: <ul style="list-style-type: none">• DATE (cast to TIMESTAMP)• TIMESTAMP• TIMESTAMPTZ• INTERVAL

Field Values

CENTURY	The century number. The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0, you go from -1 to 1.
DAY	The day (of the month) field (1-31).
DECADE	The year field divided by 10.
DOQ	The day within the current quarter. DOQ recognizes leap year days.
DOW	Zero-based day of the week, where Sunday=0. Note: EXTRACT's day of week numbering differs from the function TO_CHAR .

DOY	The day of the year (1–365/366)
EPOCH	Specifies to return one of the following: <ul style="list-style-type: none"> • For DATE and TIMESTAMP values: the number of seconds before or since 1970-01-01 00:00:00-00 (if before, a negative number). • For INTERVAL values, the total number of seconds in the interval.
HOUR	The hour field (0–23).
ISODOW	The ISO day of the week , an integer between 1 and 7 where Monday is 1.
ISOWEEK	The ISO week of the year, an integer between 1 and 53.
ISOYEAR	The ISO year .
MICROSECONDS	The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds.
MILLENNIUM	The millennium number, where the first millennium is 1 and each millenium starts on 01-01-y001. For example, millennium 2 starts on 01-01-1001.
MILLISECONDS	The seconds field, including fractional parts, multiplied by 1000. This includes full seconds.
MINUTE	The minutes field (0 - 59).
MONTH	For TIMESTAMP values, the number of the month within the year (1 - 12) ; for interval values the number of months, modulo 12 (0 - 11).
QUARTER	The calendar quarter of the specified date as an integer, where the January-March quarter is 1, valid only for TIMESTAMP values.
SECOND	The seconds field, including fractional parts, 0–59, or 0-60 if the operating system implements leap seconds.
TIME_ZONE	The time zone offset from UTC, in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.
TIMEZONE_HOUR	The hour component of the time zone offset.
TIMEZONE_	The minute component of the time zone offset.

MINUTE	
WEEK	The number of the week of the calendar year that the day is in.
YEAR	The year field. There is no 0 AD, so subtract BC years from AD years accordingly.

Examples

Extract the day of the week and day in quarter from the current TIMESTAMP:

```
=> SELECT CURRENT_TIMESTAMP AS NOW;
      NOW
-----
2016-05-03 11:36:08.829004-04
(1 row)
=> SELECT EXTRACT (DAY FROM CURRENT_TIMESTAMP);
      date_part
-----
              3
(1 row)
=> SELECT EXTRACT (DOQ FROM CURRENT_TIMESTAMP);
      date_part
-----
              33
(1 row)
```

Extract the timezone hour from the current time:

```
=> SELECT CURRENT_TIMESTAMP;
      ?column?
-----
2016-05-03 11:36:08.829004-04
(1 row)
=> SELECT EXTRACT(TIMEZONE_HOUR FROM CURRENT_TIMESTAMP);
      date_part
-----
              -4
(1 row)
```

Extract the number of seconds since 01-01-1970 00:00:

```
=> SELECT EXTRACT(EPOCH FROM '2001-02-16 20:38:40-08'::TIMESTAMP_TZ);
      date_part
-----
982384720.000000
(1 row)
```

Extract the number of seconds between 01-01-1970 00:00 and 5 days 3 hours before:

```
=> SELECT EXTRACT(EPOCH FROM -'5 days 3 hours'::INTERVAL);
       date_part
-----
-442800.000000
(1 row)
```

Convert the results from the last example to a `TIMESTAMP`:

```
=> SELECT 'EPOCH'::TIMESTAMPTZ -442800 * '1 second'::INTERVAL;
       ?column?
-----
1969-12-26 16:00:00-05
(1 row)
```

GETDATE

Returns the current statement's start date and time as a `TIMESTAMP` value. This function is identical to [SYSDATE](#).

`GETDATE` uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers. Internally, `GETDATE` converts [STATEMENT_TIMESTAMP](#) from `TIMESTAMPTZ` to `TIMESTAMP`.

Behavior Type

Stable

Syntax

```
GETDATE()
```

Example

```
=> SELECT GETDATE();
       GETDATE
-----
2011-03-07 13:21:29.497742
(1 row)
```

See Also

[Date/Time Expressions](#)

GETUTCDATE

Returns the current statement's start date and time as a `TIMESTAMP` value.

`GETUTCDATE` uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers. Internally, `GETUTCDATE` converts [STATEMENT_TIMESTAMP](#) at TIME ZONE 'UTC'.

Behavior Type

Stable

Syntax

```
GETUTCDATE()
```

Example

```
=> SELECT GETUTCDATE();
      GETUTCDATE
-----
2011-03-07 20:20:26.193052
(1 row)
```

See Also

- [Date/Time Expressions](#)

HOUR

Returns the hour portion of the specified date as an integer, where 0 is 00:00 to 00:59.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

`HOUR(date)`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>• <code>INTERVAL</code>
-------------	--

Examples

```
=> SELECT HOUR (TIMESTAMP 'sep 22, 2011 12:34');
HOUR
-----
12
(1 row)
=> SELECT HOUR (INTERVAL '35 12:34');
HOUR
-----
12
(1 row)
=> SELECT HOUR ('12:34');
HOUR
-----
12
(1 row)
```

ISFINITE

Tests for the special `TIMESTAMP` constant `INFINITY` and returns a value of type `BOOLEAN`.

Behavior Type

Immutable

Syntax

```
ISFINITE ( timestamp )
```

Parameters

<i>timestamp</i>	Expression of type <code>TIMESTAMP</code>
------------------	---

Examples

```
SELECT ISFINITE(TIMESTAMP '2009-02-16 21:28:30');
ISFINITE
-----
t
(1 row)

SELECT ISFINITE(TIMESTAMP 'INFINITY');
ISFINITE
-----
f
(1 row)
```

JULIAN_DAY

Returns the integer value of the specified day according to the Julian calendar, where day 1 is the first day of the Julian period, January 1, 4713 BC (on the Gregorian calendar, November 24, 4714 BC).

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
JULIAN_DAY ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	--

Example

```
=> SELECT JULIAN_DAY (DATE 'MARCH 15, 0044 BC');
JULIAN_DAY
-----
      1705428
(1 row)

=> SELECT JULIAN_DAY (DATE '2001-01-01');
JULIAN_DAY
-----
      2451911
(1 row)
```

LAST_DAY

Returns the last day of the month in the specified date.

Behavior Type

- Immutable if the specified is a `TIMESTAMP` or `DATE`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
LAST_DAY ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	---

Examples

The following example returns the last day of February as 29 because 2016 is a leap year:

```
=> SELECT LAST_DAY('2016-02-28 23:30 PST') "Last Day";
   Last Day
-----
2016-02-29
(1 row)
```

The following example returns the last day of February in a non-leap year:

```
> SELECT LAST_DAY('2017/02/03') "Last";
   Last
-----
2017-02-28
(1 row)
```

The following example returns the last day of March, after converting the string value to the specified `DATE` type:

```
=> SELECT LAST_DAY('2003/03/15') "Last";
       Last
-----
2012-03-31
(1 row)
```

LOCALTIME

Returns a value of type `TIME` that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to `LOCALTIME` within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

```
LOCALTIME [ ( precision ) ]
```

Parameters

<i>precision</i>	Rounds the result to the specified number of fractional digits in the seconds field.
------------------	--

Example

```
=> CREATE TABLE t1 (a int, b int);
CREATE TABLE

=> INSERT INTO t1 VALUES (1,2);
OUTPUT
-----
      1
(1 row)

=> SELECT LOCALTIME time;
       time
-----
15:03:14.595296
```

```
(1 row)

=> INSERT INTO t1 VALUES (3,4);
OUTPUT
-----
      1
(1 row)

=> SELECT LOCALTIME;
      time
-----
15:03:14.595296
(1 row)

=> COMMIT;
COMMIT
=> SELECT LOCALTIME;
      time
-----
15:03:49.738032
(1 row)
```

LOCALTIMESTAMP

Returns a value of type `TIMESTAMP` that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to `LOCALTIMESTAMP` within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

```
LOCALTIMESTAMP [ ( precision ) ]
```

Parameters

<i>precision</i>	Rounds the result to the specified number of fractional digits in the seconds field.
------------------	--

Example

```
=> CREATE TABLE t1 (a int, b int);
CREATE TABLE

=> INSERT INTO t1 VALUES (1,2);
OUTPUT
-----
      1
(1 row)

=> SELECT CURRENT_TIMESTAMP(2) timestamp;
      timestamp
-----
2016-12-07 15:19:12.34-05
(1 row)

=> INSERT INTO t1 VALUES (3,4);
OUTPUT
-----
      1
(1 row)

=> SELECT CURRENT_TIMESTAMP(2) timestamp;
      timestamp
-----
2016-12-07 15:19:12.34-05
(1 row)

=> COMMIT;
COMMIT
=> SELECT CURRENT_TIMESTAMP(2) timestamp;
      timestamp
-----
2016-12-07 15:19:13.89-05
(1 row)
```

MICROSECOND

Returns the microsecond portion of the specified date as an integer.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `INTERVAL`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMP TZ`

Syntax

`MICROSECOND (date)`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• TIMESTAMP• TIMESTAMPTZ• INTERVAL
-------------	--

Example

```
=> SELECT MICROSECOND (TIMESTAMP 'Sep 22, 2011 12:34:01.123456');
MICROSECOND
-----
          123456
(1 row)
```

MIDNIGHT_SECONDS

Within the specified date, returns the number of seconds between midnight and the date's time portion.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

`MIDNIGHT_SECONDS (date)`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ
-------------	--

Examples

Get the number of seconds since midnight:

```
=> SELECT MIDNIGHT_SECONDS(CURRENT_TIMESTAMP);
MIDNIGHT_SECONDS
-----
                36480
(1 row)
```

Get the number of seconds between midnight and noon on March 3 2016:

```
=> SELECT MIDNIGHT_SECONDS('3-3-2016 12:00'::TIMESTAMP);
MIDNIGHT_SECONDS
-----
                43200
(1 row)
```

MINUTE

Returns the minute portion of the specified date as an integer.

Behavior Type

- Immutable if the specified date is a [TIMESTAMP](#), [DATE](#), [VARCHAR](#) or [INTERVAL](#)
- Stable if the specified date is a [TIMESTAMPTZ](#)

Syntax

MINUTE (*date*)

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ• INTERVAL
-------------	---

Example

```
=> SELECT MINUTE('12:34:03.456789');
MINUTE
-----
      34
(1 row)
=>SELECT MINUTE (TIMESTAMP 'sep 22, 2011 12:34');
MINUTE
-----
      34
(1 row)
=> SELECT MINUTE(INTERVAL '35 12:34:03.456789');
MINUTE
-----
      34
(1 row)
```

MONTH

Returns the month portion of the specified date as an integer.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, `VARCHAR` or `INTERVAL`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
QUARTER ( date )
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>• <code>INTERVAL</code>
-------------	--

Examples

```
=> SELECT MONTH('6-9');
MONTH
-----
     9
(1 row)
=> SELECT MONTH (TIMESTAMP 'sep 22, 2011 12:34');
MONTH
-----
     9
(1 row)
=> SELECT MONTH(INTERVAL '2-35' year to month);
MONTH
-----
    11
(1 row)
```

MONTHS_BETWEEN

Returns the number of months between two dates. MONTHS_BETWEEN can return an integer or a FLOAT8:

- Integer: The day portions of *date1* and *date2* are the same, and neither date is the last day of the month. MONTHS_BETWEEN also returns an integer if both dates in *date1* and *date2* are the last days of their respective months. For example, MONTHS_BETWEEN calculates the difference between April 30 and March 31 as 1 month.
- FLOAT8: The day portions of *date1* and *date2* are different and one or both dates are not the last day of their respective months. For example, the difference between April 2 and March 1 is 1.03225806451613. To calculate month fractions, MONTHS_BETWEEN assumes all months contain 31 days.

MONTHS_BETWEEN disregards timestamp time portions.

Behavior Type

- Immutable if both date arguments are of data type TIMESTAMP or DATE
- Stable if either date is a TIMESTAMPTZ

Syntax

```
MONTHS_BETWEEN ( date1 , date2 );
```

Parameters

<i>date1</i> <i>date2</i>	<p>Specify the dates to evaluate where <i>date1</i> and <i>date2</i> evaluate to one of the following data types:</p> <ul style="list-style-type: none">• DATE• TIMESTAMP• TIMESTAMPTZ <p>If <i>date1</i> < <i>date2</i>, MONTHS_BETWEEN returns a negative value.</p>
------------------------------	---

Examples

Return the number of months between April 7 2016 and January 7 2015:

```
=> SELECT MONTHS_BETWEEN ('04-07-16'::TIMESTAMP, '01-07-15'::TIMESTAMP);
MONTHS_BETWEEN
-----
                15
(1 row)
```

Return the number of months between March 31 2016 and February 28 2016 (MONTHS_BETWEEN assumes both months contain 31 days):

```
=> SELECT MONTHS_BETWEEN ('03-31-16'::TIMESTAMP, '02-28-16'::TIMESTAMP);
MONTHS_BETWEEN
-----
1.09677419354839
(1 row)
```

Return the number of months between March 31 2016 and February 29 2016:

```
=> SELECT MONTHS_BETWEEN ('03-31-16'::TIMESTAMP, '02-29-16'::TIMESTAMP);
MONTHS_BETWEEN
-----
                1
(1 row)
```

NEW_TIME

Converts a timestamp value from one time zone to another and returns a `TIMESTAMP`.

Behavior Type

Immutable

Syntax

```
NEW_TIME( 'timestamp' , 'timezone1' , 'timezone2')
```

Parameters

<i>timestamp</i>	<p>The timestamp to convert, conforms to one of the following formats:</p> <ul style="list-style-type: none">• TIMESTAMP/TIMESTAMPTZ• DATE• Character string that can be converted to a <code>TIMESTAMP</code>— for example, <code>May 24, 2012 10:00</code>.
<i>timezone1</i> <i>timezone2</i>	<p>Specify the source and target timezones, one of the strings defined in <code>/opt/vertica/share/timezonesets</code>. For example:</p> <ul style="list-style-type: none">• GMT: Greenwich Mean Time• AST / ADT: Atlantic Standard/Daylight Time• EST / EDT: Eastern Standard/Daylight Time• CST / CDT: Central Standard/Daylight Time• MST / MDT: Mountain Standard/Daylight Time• PST / PDT: Pacific Standard/Daylight Time

Examples

Convert the specified time from Eastern Standard Time (EST) to Pacific Standard Time (PST):

```
=> SELECT NEW_TIME('05-24-12 13:48:00', 'EST', 'PST');
       NEW_TIME
-----
2012-05-24 10:48:00
(1 row)
```

Convert 1:00 AM January 2012 from EST to PST:

```
=> SELECT NEW_TIME('01-01-12 01:00:00', 'EST', 'PST');
       NEW_TIME
-----
```

```
2011-12-31 22:00:00
(1 row)
```

Convert the current time EST to PST:

```
=> SELECT NOW();
      NOW
-----
2016-12-09 10:30:36.727307-05
(1 row)

=> SELECT NEW_TIME('NOW', 'EDT', 'CDT');
      NEW_TIME
-----
2016-12-09 09:30:36.727307
(1 row)
```

The following example returns the year 45 before the Common Era in Greenwich Mean Time and converts it to Newfoundland Standard Time:

```
=> SELECT NEW_TIME('April 1, 45 BC', 'GMT', 'NST')::DATE;
      NEW_TIME
-----
0045-03-31 BC
(1 row)
```

NEXT_DAY

Returns the date of the first instance of a particular day of the week that follows the specified date.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
NEXT_DAY( 'date', 'day-string')
```

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ
<i>day-string</i>	The day of the week to process, a CHAR or VARCHAR string or character constant. Supply the full English name such as Tuesday, or any conventional abbreviation, such as Tue or Tues. <i>day-string</i> is not case sensitive and trailing spaces are ignored.

Examples

Get the date of the first Monday that follows April 29 2016:

```
=> SELECT NEXT_DAY('4-29-2016'::TIMESTAMP, 'Monday') "NEXT DAY" ;
      NEXT DAY
-----
2016-05-02
(1 row)
```

Get the first Tuesday that follows today:

```
SELECT NEXT_DAY(CURRENT_TIMESTAMP, 'tues') "NEXT DAY" ;
      NEXT DAY
-----
2016-05-03
(1 row)
```

NOW [Date/Time]

Returns a value of type `TIMESTAMP WITH TIME ZONE` representing the start of the current transaction. `NOW` is equivalent to [CURRENT_TIMESTAMP](#) except that it does not accept a precision parameter.

The return value does not change during the transaction. Thus, multiple calls to `CURRENT_TIMESTAMP` within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

`NOW()`

Example

```
=> CREATE TABLE t1 (a int, b int);
CREATE TABLE
=> INSERT INTO t1 VALUES (1,2);
OUTPUT
-----
      1
(1 row)

=> SELECT NOW();
      NOW
-----
2016-12-09 13:00:08.74685-05
(1 row)

=> INSERT INTO t1 VALUES (3,4);
OUTPUT
-----
      1
(1 row)

=> SELECT NOW();
      NOW
-----
2016-12-09 13:00:08.74685-05
(1 row)

=> COMMIT;
COMMIT
dbadmin=> SELECT NOW();
      NOW
-----
2016-12-09 13:01:31.420624-05
(1 row)
```

OVERLAPS

Evaluates two time periods and returns true when they overlap, false otherwise.

Behavior Type

- Stable when `TIMESTAMP` and `TIMESTAMPTZ` are both used, or when `TIMESTAMPTZ` is used with `INTERVAL`
- Immutable otherwise

Syntax

```
( start, end ) OVERLAPS ( start, end )  
( start, interval ) OVERLAPS ( start, interval )
```

Parameters

<i>start</i>	DATE, TIME, or TIMESTAMP/TIMESTAMPTZ value that specifies the beginning of a time period.
<i>end</i>	DATE, TIME, or TIMESTAMP/TIMESTAMPTZ value that specifies the end of a time period.
<i>interval</i>	Value that specifies the length of the time period.

Examples

Evaluate whether date ranges Feb 16 - Dec 21, 2016 and Oct 10 2008 - Oct 3 2016 overlap:

```
=> SELECT (DATE '2016-02-16', DATE '2016-12-21') OVERLAPS (DATE '2008-10-30', DATE '2016-10-30');  
overlaps  
-----  
t  
(1 row)
```

Evaluate whether date ranges Feb 16 - Dec 21, 2016 and Jan 01 - Oct 30 2008 - Oct 3, 2016 overlap:

```
=> SELECT (DATE '2016-02-16', DATE '2016-12-21') OVERLAPS (DATE '2008-01-30', DATE '2008-10-30');
overlaps
-----
f
(1 row)
```

Evaluate whether date range Feb 02 2016 + 1 week overlaps with date range Oct 16 2016 - 8 months:

```
=> SELECT (DATE '2016-02-16', INTERVAL '1 week') OVERLAPS (DATE '2016-10-16', INTERVAL '-8 months');
overlaps
-----
t
(1 row)
```

QUARTER

Returns calendar quarter of the specified date as an integer, where the January-March quarter is 1.

Syntax

```
QUARTER ( date )
```

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`.
- Stable if the specified date is a `TIMESTAMPTZ`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	--

Examples

```
=> SELECT QUARTER (TIMESTAMP 'sep 22, 2011 12:34');
   QUARTER
-----
          3
(1 row)
```

ROUND

Rounds the specified date or time. If you omit the precision argument, ROUND rounds to day (DD) precision.

Behavior Type

- Immutable if the target date is a `TIMESTAMP` or `DATE`
- Stable if the target date is a `TIMESTAMPTZ`

Syntax

```
ROUND( rounding-target [, 'precision' ] )
```

Parameters

<i>rounding-target</i>	An expression that evaluates to one of the following data types: <ul style="list-style-type: none">• <code>DATE</code>• <code>TIMESTAMP/TIMESTAMPTZ</code>• <code>TIMESTAMPTZ</code>
<i>precision</i>	A string constant that specifies precision for the rounded value, one of the following: <ul style="list-style-type: none">• Century: <code>CC</code> <code>SCC</code>

- **Year:** SYYY | YYYY | YEAR | YYY | YY | Y
- **ISO Year:** IYYY | IYY | IY | I
- **Quarter:** Q
- **Month:** MONTH | MON | MM | RM
- **Same weekday as first day of year:** WW
- **Same weekday as first day of ISO year:** IW
- **Same weekday as first day of month:** W
- **Day (default):** DDD | DD | J
- **First weekday:** DAY | DY | D
- **Hour:** HH | HH12 | HH24
- **Minute:** MI
- **Second:** SS

Note: Hour, minute, and second rounding is not supported by DATE expressions.

Examples

Round to the nearest hour:

```
=> SELECT ROUND(CURRENT_TIMESTAMP, 'HH');
      ROUND
-----
2016-04-28 15:00:00
(1 row)
```

Round to the nearest month:

```
=> SELECT ROUND('9-22-2011 12:34:00'::TIMESTAMP, 'MM');
      ROUND
-----
2011-10-01 00:00:00
(1 row)
```

See Also

[TIMESTAMP_ROUND](#)

SECOND

Returns the seconds portion of the specified date as an integer.

Syntax

```
SECOND ( date )
```

Behavior Type

Immutable, except for TIMESTAMPTZ arguments where it is stable.

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• TIMESTAMP• TIMESTAMPTZ• INTERVAL
-------------	--

Examples

```
=> SELECT SECOND ('23:34:03.456789');
SECOND
-----
      3
(1 row)
=> SELECT SECOND (TIMESTAMP 'sep 22, 2011 12:34');
SECOND
-----
```

```
0
(1 row)
=> SELECT SECOND (INTERVAL '35 12:34:03.456789');
SECOND
-----
3
(1 row)
```

STATEMENT_TIMESTAMP

Similar to [TRANSACTION_TIMESTAMP](#), returns a value of type `TIMESTAMP WITH TIME ZONE` that represents the start of the current statement.

The return value does not change during statement execution. Thus, different stages of statement execution always have the same timestamp.

Behavior Type

Stable

Syntax

```
STATEMENT_TIMESTAMP()
```

Example

```
=> SELECT foo, bar FROM (SELECT STATEMENT_TIMESTAMP() AS foo)foo, (SELECT STATEMENT_TIMESTAMP() as
bar)bar;
      foo                |      bar
-----+-----
2016-12-07 14:55:51.543988-05 | 2016-12-07 14:55:51.543988-05
(1 row)
```

See Also

- [CLOCK_TIMESTAMP](#)
- [TRANSACTION_TIMESTAMP](#)

SYSDATE

Returns the current statement's start date and time as a `TIMESTAMP` value. This function is identical to [GETDATE](#).

`SYSDATE` uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers. Internally, `GETDATE` converts [STATEMENT_TIMESTAMP](#) from `TIMESTAMPTZ` to `TIMESTAMP`.

Behavior Type

Stable

Syntax

`SYSDATE()`

Note: You can call this function with no parentheses.

Example

```
=> SELECT SYSDATE;  
      sysdate  
-----  
2016-12-12 06:11:10.699642  
(1 row)
```

See Also

[Date/Time Expressions](#)

TIME_SLICE

Aggregates data by different fixed-time intervals and returns a rounded-up input `TIMESTAMP` value to a value that corresponds with the start or end of the time slice interval.

Given an input `TIMESTAMP` value such as `2000-10-28 00:00:01`, the start time of a 3-second time slice interval is `2000-10-28 00:00:00`, and the end time of the same time slice is `2000-10-28 00:00:03`.

Behavior Type

Immutable

Syntax

```
TIME_SLICE( expression, slice-length [, 'time--unit' [, 'start-or-end' ] ] )
```

Parameters

<i>expression</i>	<p>One of the following:</p> <ul style="list-style-type: none">• Column of type <code>TIMESTAMP</code>• String constant that can be parsed into a <code>TIMESTAMP</code> value. For example: <pre>'2004-10-19 10:23:54'</pre> <p>Vertica evaluates <i>expression</i> on each row.</p>
<i>slice-length</i>	A positive integer that specifies the slice length.
<i>time--unit</i>	<p>Time unit of the slice, one of the following:</p> <ul style="list-style-type: none">• <code>HOUR</code>• <code>MINUTE</code>• <code>SECOND</code> (default)• <code>MILLISECOND</code>• <code>MICROSECOND</code>
<i>'start-or-end'</i>	Specifies whether the returned value corresponds to the start or end time with one of the following strings:

- START (default)
- END

Note: This parameter can be included only if you also supply a non-null *time--unit* argument.

Null Argument Handling

TIME_SLICE handles null arguments as follows:

- TIME_SLICE returns an error when any one of *slice-length*, *time--unit*, or *start-or-end* parameters is null.
- If *expression* is null and *slice-length*, *time--unit*, or *start-or-end* contain legal values, TIME_SLICE returns a NULL value instead of an error.

Usage

The following command returns the (default) start time of a 3-second time slice:

```
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3);
      TIME_SLICE
-----
2009-09-19 00:00:00
(1 row)
```

The following command returns the end time of a 3-second time slice:

```
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'SECOND', 'END');
      TIME_SLICE
-----
2009-09-19 00:00:03
(1 row)
```

This command returns results in milliseconds, using a 3-second time slice:

```
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'ms');
      TIME_SLICE
-----
2009-09-19 00:00:00.999
(1 row)
```

This command returns results in microseconds, using a 9-second time slice:

```
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'us');
       TIME_SLICE
-----
2009-09-19 00:00:00.999999
(1 row)
```

The next example uses a 3-second interval with an input value of '00:00:01'. To focus specifically on seconds, the example omits date, though all values are implied as being part of the timestamp with a given input of '00:00:01':

- '00:00:00' is the start of the 3-second time slice
- '00:00:03' is the end of the 3-second time slice.
- '00:00:03' is also the start of the *second* 3-second time slice. In time slice boundaries, the end value of a time slice does not belong to that time slice; it starts the next one.

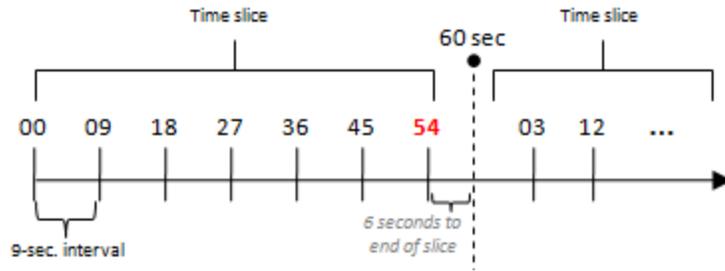


When the time slice interval is not a factor of 60 seconds, such as a given slice length of 9 in the following example, the slice does not always start or end on 00 seconds:

```
=> SELECT TIME_SLICE('2009-02-14 20:13:01', 9);
       TIME_SLICE
-----
2009-02-14 20:12:54
(1 row)
```

This is expected behavior, as the following properties are true for all time slices:

- Equal in length
- Consecutive (no gaps between them)
- Non-overlapping



To force the above example ('2009-02-14 20:13:01') to start at '2009-02-14 20:13:00', adjust the output timestamp values so that the remainder of 54 counts up to 60:

```
=> SELECT TIME_SLICE('2009-02-14 20:13:01', 9)+'6 seconds'::INTERVAL AS time;
       time
-----
2009-02-14 20:13:00
(1 row)
```

Alternatively, you could use a different slice length, which is divisible by 60, such as 5:

```
=> SELECT TIME_SLICE('2009-02-14 20:13:01', 5);
       TIME_SLICE
-----
2009-02-14 20:13:00
(1 row)
```

A TIMESTAMPTZ value is implicitly cast to TIMESTAMP. For example, the following two statements have the same effect.

```
=> SELECT TIME_SLICE('2009-09-23 11:12:01'::timestamptz, 3);
       TIME_SLICE
-----
2009-09-23 11:12:00
(1 row)

=> SELECT TIME_SLICE('2009-09-23 11:12:01'::timestamptz::timestamp, 3);
       TIME_SLICE
-----
2009-09-23 11:12:00
(1 row)
```

Examples

You can use the SQL analytic functions `FIRST_VALUE` and `LAST_VALUE` to find the first/last price within each time slice group (set of rows belonging to the same time slice). This structure could be useful if you want to sample input data by choosing one row from each time slice group.

```
=> SELECT date_key, transaction_time, sales_dollar_amount, TIME_SLICE(DATE '2000-01-01' + date_key +
transaction_time, 3),
FIRST_VALUE(sales_dollar_amount)
OVER (PARTITION BY TIME_SLICE(DATE '2000-01-01' + date_key + transaction_time, 3)
ORDER BY DATE '2000-01-01' + date_key + transaction_time) AS first_value
FROM store.store_sales_fact
LIMIT 20;
```

date_key	transaction_time	sales_dollar_amount	time_slice	first_value
1	00:41:16	164	2000-01-02 00:41:15	164
1	00:41:33	310	2000-01-02 00:41:33	310
1	15:32:51	271	2000-01-02 15:32:51	271
1	15:33:15	419	2000-01-02 15:33:15	419
1	15:33:44	193	2000-01-02 15:33:42	193
1	16:36:29	466	2000-01-02 16:36:27	466
1	16:36:44	250	2000-01-02 16:36:42	250
2	03:11:28	39	2000-01-03 03:11:27	39
3	03:55:15	375	2000-01-04 03:55:15	375
3	11:58:05	369	2000-01-04 11:58:03	369
3	11:58:24	174	2000-01-04 11:58:24	174
3	11:58:52	449	2000-01-04 11:58:51	449
3	19:01:21	201	2000-01-04 19:01:21	201
3	22:15:05	156	2000-01-04 22:15:03	156
4	13:36:57	-125	2000-01-05 13:36:57	-125
4	13:37:24	-251	2000-01-05 13:37:24	-251
4	13:37:54	353	2000-01-05 13:37:54	353
4	13:38:04	426	2000-01-05 13:38:03	426
4	13:38:31	209	2000-01-05 13:38:30	209
5	10:21:24	488	2000-01-06 10:21:24	488

(20 rows)

TIME_SLICE rounds the transaction time to the 3-second slice length.

The following example uses the [analytic \(window\) OVER clause](#) to return the last trading price (the last row ordered by TickTime) in each 3-second time slice partition:

```
=> SELECT DISTINCT TIME_SLICE(TickTime, 3), LAST_VALUE(price)OVER (PARTITION BY TIME_SLICE(TickTime,
3)
ORDER BY TickTime ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING);
```

Note: If you omit the windowing clause from an analytic clause, LAST_VALUE defaults to RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. Results can seem non-intuitive, because instead of returning the value from the bottom of the current partition, the function returns the bottom of the *window*, which continues to change along with the current input row that is being processed. For more information, see [Time Series Analytics](#) and [SQL Analytics](#) in Analyzing Data.

In the next example, FIRST_VALUE is evaluated once for each input record and the data is sorted by ascending values. Use SELECT DISTINCT to remove the duplicates and return only one output record per TIME_SLICE:

```
=> SELECT DISTINCT TIME_SLICE(TickTime, 3), FIRST_VALUE(price)OVER (PARTITION BY TIME_SLICE(TickTime, 3)
ORDER BY TickTime ASC)
FROM tick_store;
  TIME_SLICE      | ?column?
-----+-----
2009-09-21 00:00:06 |    20.00
2009-09-21 00:00:09 |    30.00
2009-09-21 00:00:00 |    10.00
(3 rows)
```

The information output by the above query can also return MIN, MAX, and AVG of the trading prices within each time slice.

```
=> SELECT DISTINCT TIME_SLICE(TickTime, 3),FIRST_VALUE(Price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)
ORDER BY TickTime ASC),
  MIN(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)),
  MAX(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3)),
  AVG(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3))
FROM tick_store;
```

See Also

- [Aggregate Functions](#)
- [FIRST_VALUE \[Analytic\]](#)
- [LAST_VALUE \[Analytic\]](#)
- [TIMESERIES Clause](#)
- [TS_FIRST_VALUE](#)
- [TS_LAST_VALUE](#)
- [Using Time Zones With Vertica](#)

TIMEOFDAY

Returns the wall-clock time as a text string. Function results advance during transactions.

Behavior Type

Volatile

Syntax

TIMEOFDAY()

Example

```
=> SELECT TIMEOFDAY();
      TIMEOFDAY
-----
Mon Dec 12 08:18:01.022710 2016 EST
(1 row)
```

TIMESTAMPADD

Adds the specified number of intervals to a `TIMESTAMP` or `TIMESTAMPTZ` value and returns a result of the same data type.

Behavior Type

- Immutable if the input date is a `TIMESTAMP`
- Stable if the input date is a `TIMESTAMPTZ`

Syntax

`TIMESTAMPADD (datepart, count, start-date);`

Parameters

<i>datepart</i>	<p>Specifies the type of time intervals that <code>TIMESTAMPADD</code> adds to the specified start date. If <i>datepart</i> is an expression, it must be enclosed in parentheses:</p> <pre>TIMESTAMPADD(<i>expression</i>, <i>interval</i>, <i>start</i>;</pre> <p><i>datepart</i> must evaluate to one of the following string literals, either quoted or unquoted:</p>
-----------------	--

	<ul style="list-style-type: none"> • year yy yyyy • quarter qq q • month mm m • day dayofyear dd d dy y • week wk ww • hour hh • minute mi n • second ss s • millisecond ms • microsecond mcs us
<i>count</i>	Integer or integer expression that specifies the number of <i>datepart</i> intervals to add to <i>start-date</i> .
<i>start-date</i>	TIMESTAMP or TIMESTAMPTZ value.

Examples

Add two months to the current date:

```
=> SELECT CURRENT_TIMESTAMP AS Today;
      Today
-----
2016-05-02 06:56:57.923045-04
(1 row)

=> SELECT TIMESTAMPADD (MONTH, 2, (CURRENT_TIMESTAMP)) AS TodayPlusTwoMonths;;
      TodayPlusTwoMonths
-----
2016-07-02 06:56:57.923045-04
(1 row)
```

Add 14 days to the beginning of the current month:

```
=> SELECT TIMESTAMPADD (DD, 14, (SELECT TRUNC((CURRENT_TIMESTAMP), 'MM')));
      timestampadd
-----
2016-05-15 00:00:00
(1 row)
```

TIMESTAMPDIFF

Returns the time span between two `TIMESTAMP` or `TIMESTAMPTZ` values, in the intervals specified. `TIMESTAMPDIFF` excludes the start date in its calculation.

Behavior Type

- Immutable if start and end dates are `TIMESTAMP`
- Stable if start and end dates are `TIMESTAMPTZ`

Syntax

```
TIMESTAMPDIFF ( datepart, start, end );
```

Parameters

<i>datepart</i>	<p>Specifies the type of date or time intervals that <code>TIMESTAMPDIFF</code> returns. If <i>datepart</i> is an expression, it must be enclosed in parentheses:</p> <pre>TIMESTAMPDIFF(<i>expression</i>), <i>start</i>, <i>end</i> ;</pre> <p><i>datepart</i> must evaluate to one of the following string literals, either quoted or unquoted:</p> <ul style="list-style-type: none">• year yy yyyy• quarter qq q• month mm m• day dayofyear dd d dy y• week wk ww• hour hh• minute mi n• second ss s
-----------------	--

	<ul style="list-style-type: none">• millisecond ms• microsecond mcs us
<i>start, end</i>	<p>Specify the start and end dates, where <i>start</i> and <i>end</i> evaluate to one of the following data types:</p> <ul style="list-style-type: none">• TIMESTAMP/TIMESTAMP TZ• TIMESTAMP TZ <p>If <i>end</i> < <i>start</i>, <code>TIMESTAMPDIFF</code> returns a negative value.</p>

Date Part Intervals

`TIMESTAMPDIFF` uses the *datepart* argument to calculate the number of intervals between two dates, rather than the actual amount of time between them. For detailed information, see [DATEDIFF](#).

Examples

```
=> SELECT TIMESTAMPDIFF (YEAR, '1-1-2006 12:34:00', '1-1-2008 12:34:00');
timestampdiff
-----
                2
(1 row)
```

See Also

[DATEDIFF](#)

TIMESTAMP_ROUND

Rounds the specified `TIMESTAMP`. If you omit the precision argument, `TIMESTAMP_ROUND` rounds to day (DD) precision.

Behavior Type

- Immutable if the target date is a `TIMESTAMP`
- Stable if the target date is a `TIMESTAMPTZ`

Syntax

```
TIMESTAMP_ROUND ( rounding-target [, 'precision' ] )
```

Parameters

<i>rounding-target</i>	An expression that evaluates to one of the following data types: <ul style="list-style-type: none">• TIMESTAMP/TIMESTAMPTZ• TIMESTAMPTZ
<i>precision</i>	A string constant that specifies precision for the rounded value, one of the following: <ul style="list-style-type: none">• Century: CC SCC• Year: SYYY YYYY YEAR YYY YY Y• ISO Year: IYYY IYY IY I• Quarter: Q• Month: MONTH MON MM RM• Same weekday as first day of year: WW• Same weekday as first day of ISO year: IW• Same weekday as first day of month: W• Day (default): DDD DD J• First weekday: DAY DY D

- **Hour:** HH | HH12 | HH24
- **Minute:** MI
- **Second:** SS

Note: Hour, minute, and second rounding is not supported by DATE expressions.

Examples

Round to the nearest hour:

```
=> SELECT TIMESTAMP_ROUND(CURRENT_TIMESTAMP, 'HH');  
        ROUND  
-----  
2016-04-28 15:00:00  
(1 row)
```

Round to the nearest month:

```
=> SELECT TIMESTAMP_ROUND('9-22-2011 12:34:00'::TIMESTAMP, 'MM');  
        ROUND  
-----  
2011-10-01 00:00:00  
(1 row)
```

See Also

[ROUND](#)

TIMESTAMP_TRUNC

Truncates the specified `TIMESTAMP`. If you omit the precision argument, `TIMESTAMP_TRUNC` truncates to day (DD) precision.

Behavior Type

- Immutable if the target date is a `TIMESTAMP`
- Stable if the target date is a `TIMESTAMPPTZ`

Syntax

```
TIMESTAMP_TRUNC( trunc-target [, 'precision' ] )
```

Parameters

<i>trunc-target</i>	An expression that evaluates to one of the following data types: <ul style="list-style-type: none">• TIMESTAMP/TIMESTAMP TZ• TIMESTAMP TZ
<i>precision</i>	A string constant that specifies precision for the truncated value, one of the following: <ul style="list-style-type: none">• Century: CC SCC• Year: SYYY YYYY YEAR YYY YY Y• ISO Year: IYYY IYY IY I• Quarter: Q• Month: MONTH MON MM RM• Same weekday as first day of year: WW• Same weekday as first day of ISO year: IW• Same weekday as first day of month: W• Day: DDD DD J• First weekday: DAY DY D• Hour: HH HH12 HH24• Minute: MI• Second: SS <p>Note: Hour, minute, and second truncating is not supported by</p>

DATE expressions.

Examples

Truncate to the current hour:

```
=> SELECT TIMESTAMP_TRUNC(CURRENT_TIMESTAMP, 'HH');
TIMESTAMP_TRUNC
-----
2016-04-29 08:00:00
(1 row)
```

Truncate to the month:

```
=> SELECT TIMESTAMP_TRUNC('9-22-2011 12:34:00'::TIMESTAMP, 'MM');
TIMESTAMP_TRUNC
-----
2011-09-01 00:00:00
(1 row)
```

See Also

[TRUNC](#)

TRANSACTION_TIMESTAMP

Returns a value of type `TIME WITH TIMEZONE` that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to `TRANSACTION_TIMESTAMP` within the same transaction return the same timestamp.

`TRANSACTION_TIMESTAMP` is equivalent to [CURRENT_TIMESTAMP](#), except it does not accept a precision parameter.

Behavior Type

Stable

Syntax

TRANSACTION_TIMESTAMP()

Example

```
=> SELECT foo, bar FROM (SELECT TRANSACTION_TIMESTAMP() AS foo)foo, (SELECT TRANSACTION_TIMESTAMP()
as bar)bar;
      foo                |                bar
-----+-----
2016-12-12 08:18:00.988528-05 | 2016-12-12 08:18:00.988528-05
(1 row)
```

See Also

- [CLOCK_TIMESTAMP](#)
- [STATEMENT_TIMESTAMP](#)

TRUNC

Truncates the specified date or time. If you omit the precision argument, TRUNC truncates to day (DD) precision.

Behavior Type

- Immutable if the target date is a `TIMESTAMP` or `DATE`
- Stable if the target date is a `TIMESTAMPTZ`

Syntax

TRUNC(*trunc-target* [, '*precision*'])

Parameters

<i>trunc-target</i>	<p>An expression that evaluates to one of the following data types:</p> <ul style="list-style-type: none">• DATE• TIMESTAMP/TIMESTAMPTZ• TIMESTAMPTZ
<i>precision</i>	<p>A string constant that specifies precision for the truncated value, one of the following:</p> <ul style="list-style-type: none">• Century: CC SCC• Year: SYYY YYYY YEAR YYY YY Y• ISO Year: IYYY IYY IY I• Quarter: Q• Month: MONTH MON MM RM• Same weekday as first day of year: WW• Same weekday as first day of ISO year: IW• Same weekday as first day of month: W• Day (default): DDD DD J• First weekday: DAY DY D• Hour: HH HH12 HH24• Minute: MI• Second: SS <p>Note: Hour, minute, and second truncating is not supported by DATE expressions.</p>

Examples

Truncate to the current hour:

```
=> => SELECT TRUNC(CURRENT_TIMESTAMP, 'HH');
      TRUNC
-----
2016-04-29 10:00:00
(1 row)
```

Truncate to the month:

```
=> SELECT TRUNC('9-22-2011 12:34:00'::TIMESTAMP, 'MM');
      TIMESTAMP_TRUNC
-----
2011-09-01 00:00:00
(1 row)
```

See Also

[TIMESTAMP_TRUNC](#)

WEEK

Returns the week of the year for the specified date as an integer, where the first week begins on the first Sunday on or preceding January 1.

Syntax

```
WEEK ( date )
```

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPPTZ`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• VARCHAR• DATE• TIMESTAMP• TIMESTAMPTZ
-------------	--

Examples

January 2 is on Saturday, so WEEK returns 1:

```
=> SELECT WEEK ('1-2-2016'::DATE);
WEEK
-----
    1
(1 row)
```

January 3 is the second Sunday in 2016, so WEEK returns 2:

```
=> SELECT WEEK ('1-3-2016'::DATE);
WEEK
-----
    2
(1 row)
```

WEEK_ISO

Returns the week of the year for the specified date as an integer, where the first week starts on Monday and contains January 4. This function conforms with the ISO 8061 standard.

Syntax

```
WEEK_ISO ( date )
```

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPTZ`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	--

Examples

The first week of 2016 begins on Monday January 4:

```
=> SELECT WEEK_ISO ('1-4-2016'::DATE);  
WEEK_ISO  
-----  
          1  
(1 row)
```

January 3 2016 returns week 53 of the previous year (2015):

```
=> SELECT WEEK_ISO ('1-3-2016'::DATE);  
WEEK_ISO  
-----  
          53  
(1 row)
```

In 2015, January 4 is on Sunday, so the first week of 2015 begins on the preceding Monday (December 29 2014):

```
=> SELECT WEEK_ISO ('12-29-2014'::DATE);  
WEEK_ISO  
-----  
          1
```

(1 row)

YEAR

Returns an integer that represents the year portion of the specified date.

Syntax

```
YEAR( date )
```

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, `VARCHAR`, or `INTERVAL`
- Stable if the specified date is a `TIMESTAMPTZ`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>• <code>INTERVAL</code>
-------------	--

Examples

```
=> SELECT YEAR(CURRENT_DATE::DATE);  
YEAR  
-----  
2016  
(1 row)
```

See Also

[YEAR_ISO](#)

YEAR_ISO

Returns an integer that represents the year portion of the specified date. The return value is based on the ISO 8061 standard.

The first week of the ISO year is the week that contains January 4.

Syntax

```
YEAR_ISO ( date )
```

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `VARCHAR`
- Stable if the specified date is a `TIMESTAMPTZ`

Parameters

<i>date</i>	The date to process, one of the following data types: <ul style="list-style-type: none">• <code>VARCHAR</code>• <code>DATE</code>• <code>TIMESTAMP</code>• <code>TIMESTAMPTZ</code>
-------------	--

Examples

```
> SELECT YEAR_ISO(CURRENT_DATE::DATE);  
YEAR_ISO  
-----  
      2016  
(1 row)
```

See Also

[YEAR](#)

Error-handling Functions

Error-handling functions take a string and return the string when the query is executed.

THROW_ERROR

Returns a user-defined error message.

In a multi-node cluster, race conditions might cause the order of error messages to differ.

Syntax

```
THROW_ERROR ( message )
```

Parameters

<i>message</i>	The VARCHAR string to to return.
----------------	----------------------------------

Examples

Return an error message when a CASE statement is met:

```
=> CREATE TABLE pitcher_err (some_text varchar);
CREATE TABLE
=> COPY pitcher_err FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> big foo value
>> bigger foo other value
>> bar another foo value
>> \.
=> SELECT (CASE WHEN true THEN THROW_ERROR('Failure!!!') ELSE some_text END) FROM pitcher_err;
ERROR 7137:  USER GENERATED ERROR: Failure!!!
```

Return an error message when a CASE statement using [REGEXP_LIKE](#) is met:

```
=> SELECT (CASE WHEN REGEXP_LIKE(some_text, 'other') THEN THROW_ERROR('Failure at "' || some_text ||
''') END) FROM pitcher_err;
ERROR 4566:  USER GENERATED ERROR: Failure at "bar another foo value"
```

Formatting Functions

Formatting functions provide a powerful tool set for converting various data types (DATE/TIME, INTEGER, FLOATING POINT) to formatted strings and for converting from formatted strings to specific data types.

TO_BITSTRING

Returns a VARCHAR that represents the given VARBINARY value in bitstring format. This function is the inverse of [BITSTRING_TO_BINARY](#).

Behavior Type

Immutable

Syntax

```
TO_BITSTRING ( expression )
```

Parameters

<i>expression</i>	The VARCHAR string to process.
-------------------	--------------------------------

Examples

```
=> SELECT TO_BITSTRING('ab'::BINARY(2));
   to_bitstring
-----
0110000101100010
(1 row)
```

```
=> SELECT TO_BITSTRING(HEX_TO_BINARY('0x10'));
   to_bitstring
-----
00010000
(1 row)

=> SELECT TO_BITSTRING(HEX_TO_BINARY('0xF0'));
   to_bitstring
-----
11110000
(1 row)
```

See Also

[BITCOUNT](#)

TO_CHAR

Converts various date/time and numeric values into text strings.

Behavior Type

Stable

Syntax

`TO_CHAR (expression [, pattern])`

Parameters

<i>expression</i>	Specifies the value to convert, one of the following data types: <ul style="list-style-type: none">DOUBLE PRECISIONINTEGERINTERVALTIME/TIMETZTIMESTAMP/TIMESTAMPTZ
-------------------	--

<i>pattern</i>	A CHAR or VARCHAR that specifies an output pattern string. See: <ul style="list-style-type: none">• Template Patterns for Date/Time Formatting• Template Patterns for Numeric Formatting
----------------	---

Notes

- TO_CHAR(any) casts any type, except BINARY/VARBINARY, to VARCHAR.

The following example returns an error if you try to cast TO_CHAR to a binary data type:

```
=> SELECT TO_CHAR('abc'::VARBINARY);  
ERROR: cannot cast type varbinary to varchar
```

- TO_CHAR accepts TIME and TIMETZ data types as inputs if you explicitly cast TIME to TIMESTAMP and TIMETZ to TIMESTAMPTZ.

```
=> SELECT TO_CHAR(TIME '14:34:06.4', 'HH12:MI am');  
=> SELECT TO_CHAR(TIMETZ '14:34:06.4+6', 'HH12:MI am');
```

You can extract the timezone hour from TIMETZ:

```
=> SELECT EXTRACT(timezone_hour FROM TIMETZ '10:30+13:30');  
date_part  
-----  
13  
(1 row)
```

- Ordinary text is allowed in to_char templates and is output literally. You can put a substring in double quotes to force it to be interpreted as literal text even if it contains pattern key words. For example, in '"Hello Year "YYYY"', the YYYY is replaced by the year data, but the single Y in Year is not.
- TO_CHAR's day-of-the-week numbering (see the 'D' [template pattern](#)) is different from that of the EXTRACT function.
- Given an INTERVAL type, TO_CHAR formats HH and HH12 as hours in a single day, while HH24 can output hours exceeding a single day, for example, >24.
- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: '\\\"YYYY Month\\\"'

- TO_CHAR does not support the use of V combined with a decimal point. For example: 99.9V99 is not allowed.
- When rounding, the last digit of the rounded representation is selected to be even if the number is exactly half way between the two.

Examples

Expression	Result
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'Day, DD HH12:MI:SS');	'Tuesday , 06 05:39:18'
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'FMDay, FMDD HH12:MI:SS');	'Tuesday, 6 05:39:18'
SELECT TO_CHAR(TIMETz '14:34:06.4+6', 'HH12:MI am'); TO_CHAR	04:34 am
SELECT TO_CHAR(-0.1, '99.99');	' -.10'
SELECT TO_CHAR(-0.1, 'FM9.99');	'-.1'
SELECT TO_CHAR(0.1, '0.9');	' 0.1'
SELECT TO_CHAR(12, '9990999.9');	' 0012.0'
SELECT TO_CHAR(12, 'FM9990999.9');	'0012.'
SELECT TO_CHAR(485, '999');	' 485'
SELECT TO_CHAR(-485, '999');	'-485'
SELECT TO_CHAR(485, '9 9 9');	' 4 8 5'
SELECT TO_CHAR(1485, '9,999');	' 1,485'
SELECT TO_CHAR(1485, '9G999');	' 1 485'
SELECT TO_CHAR(148.5, '999.999');	' 148.500'
SELECT TO_CHAR(148.5, 'FM999.999');	'148.5'
SELECT TO_CHAR(148.5, 'FM999.990');	'148.500'
SELECT TO_CHAR(148.5, '999D999');	' 148,500'
SELECT TO_CHAR(3148.5, '9G999D999');	' 3 148,500'
SELECT TO_CHAR(-485, '999S');	'485-'
SELECT TO_CHAR(-485, '999MI');	'485-'
SELECT TO_CHAR(485, '999MI');	'485 '
SELECT TO_CHAR(485, 'FM999MI');	'485'

SELECT TO_CHAR(485, 'PL999');	'+485'
SELECT TO_CHAR(485, 'SG999');	'+485'
SELECT TO_CHAR(-485, 'SG999');	'-485'
SELECT TO_CHAR(-485, '9SG99');	'4-85'
SELECT TO_CHAR(-485, '999PR');	'<485>'
SELECT TO_CHAR(485, 'L999');	'DM 485'
SELECT TO_CHAR(485, 'RN');	' CDLXXXV'
SELECT TO_CHAR(485, 'FMRN');	'CDLXXXV'
SELECT TO_CHAR(5.2, 'FMRN');	'V'
SELECT TO_CHAR(482, '999th');	' 482nd'
SELECT TO_CHAR(485, '"Good number:"999');	'Good number: 485'
SELECT TO_CHAR(485.8, '"Pre:"999" Post:" .999');	'Pre: 485 Post: .800'
SELECT TO_CHAR(12, '99V999');	' 12000'
SELECT TO_CHAR(12.4, '99V999');	' 12400'
SELECT TO_CHAR(12.45, '99V9');	' 125'
SELECT TO_CHAR(-1234.567);	-1234.567
SELECT TO_CHAR('1999-12-25'::DATE);	1999-12-25
SELECT TO_CHAR('1999-12-25 11:31'::TIMESTAMP);	1999-12-25 11:31:00
SELECT TO_CHAR('1999-12-25 11:31 EST'::TIMESTAMPZ);	1999-12-25 11:31:00-05
SELECT TO_CHAR('3 days 1000.333 secs'::INTERVAL);	3 days 00:16:40.333

TO_DATE

Converts a string value to a DATE type.

Behavior Type

Stable

Syntax

TO_DATE (*expression* , *pattern*)

Parameters

<i>expression</i>	Specifies the string value to convert, either CHAR or VARCHAR.
<i>pattern</i>	A CHAR or VARCHAR that specifies an output pattern string. See: <ul style="list-style-type: none">Template Patterns for Date/Time FormattingTemplate Patterns for Numeric Formatting

Input Value Considerations

TO_DATE requires a CHAR or VARCHAR expression. For other input types, use [TO_CHAR](#) to perform an explicit cast to a CHAR or VARCHAR before using this function.

Notes

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: `'\\"YYYY Month\\"'`
- TO_TIMESTAMP, TO_TIMESTAMP_TZ, and TO_DATE skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:
 - TO_TIMESTAMP('2000 JUN', 'YYYY MON') is correct.
 - TO_TIMESTAMP('2000 JUN', 'FXYYYY MON') returns an error, because TO_TIMESTAMP expects one space only.
- The YYYY conversion from string to TIMESTAMP or DATE has a restriction if you use a year with more than four digits. You must use a non-digit character or template after YYYY, otherwise the year is always interpreted as four digits. For example, given the following arguments, TO_DATE interprets the five-digit year 20000 as a four-digit year:

```
=> SELECT TO_DATE('200001131', 'YYYYMMDD');  
TO_DATE  
-----
```

```
2000-01-13  
(1 row)
```

Instead, use a non-digit separator after the year. For example:

```
=> SELECT TO_DATE('20000-1131', 'YYYY-MMDD');  
      TO_DATE  
-----  
20000-12-01  
(1 row)
```

- In conversions from string to `TIMESTAMP` or `DATE`, the `CC` field is ignored if there is a `YYY`, `YYYY` or `Y,YYY` field. If `CC` is used with `YY` or `Y`, then the year is computed as $(CC-1)*100+YY$.

Examples

```
=> SELECT TO_DATE('13 Feb 2000', 'DD Mon YYYY');  
      to_date  
-----  
2000-02-13  
(1 row)
```

See Also

[Date/Time Functions](#)

TO_HEX

Returns a `VARCHAR` or `VARBINARY` representing the hexadecimal equivalent of a number. This function is the inverse of [HEX_TO_BINARY](#).

Behavior Type

Immutable

Syntax

```
TO_HEX ( number )
```

Parameters

<i>number</i>	An INTEGER or VARBINARY value to convert to hexadecimal. If you supply a VARBINARY argument, the function's return value is not preceded by 0x.
---------------	--

Examples

```
=> SELECT TO_HEX(123456789);
   TO_HEX
-----
   75bcd15
(1 row)
```

For **VARBINARY** inputs, the returned value is not preceded by 0x. For example:

```
=> SELECT TO_HEX('ab'::binary(2));
   TO_HEX
-----
     6162
(1 row)
```

TO_TIMESTAMP

Converts a string value or a UNIX/POSIX epoch value to a **TIMESTAMP** type.

Behavior Type

Stable

Syntax

```
TO_TIMESTAMP ( expression, pattern )TO_TIMESTAMP ( unix-epoch )
```

Parameters

<i>expression</i>	Specifies the string value to convert, either CHAR or VARCHAR .
<i>pattern</i>	A CHAR or VARCHAR that specifies an output pattern string. See:

	<ul style="list-style-type: none"> • Template Patterns for Date/Time Formatting • Template Patterns for Numeric Formatting
<i>unix-epoch</i>	A DOUBLE PRECISION value that specifies some number of seconds elapsed since midnight UTC of January 1, 1970, not counting leap seconds. INTEGER values are implicitly cast to DOUBLE PRECISION.

Notes

- Millisecond (MS) and microsecond (US) values in a conversion from string to TIMESTAMP are used as part of the seconds after the decimal point. For example TO_TIMESTAMP('12:3', 'SS:MS') is not 3 milliseconds, but 300, because the conversion counts it as 12 + 0.3 seconds. This means for the format SS:MS, the input values 12:3, 12:30, and 12:300 specify the same number of milliseconds. To get three milliseconds, use 12:003, which the conversion counts as 12 + 0.003 = 12.003 seconds.

Here is a more complex example: TO_TIMESTAMP('15:12:02.020.001230', 'HH:MI:SS.MS.US') is 15 hours, 12 minutes, and 2 seconds + 20 milliseconds + 1230 microseconds = 2.021230 seconds.

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: '\\\"YYYY Month\\\"'
- TO_TIMESTAMP, TO_TIMESTAMP_TZ, and TO_DATE skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:
 - TO_TIMESTAMP('2000 JUN', 'YYYY MON') is correct.
 - TO_TIMESTAMP('2000 JUN', 'FXYYYY MON') returns an error, because TO_TIMESTAMP expects one space only.
- The YYYY conversion from string to TIMESTAMP or DATE has a restriction if you use a year with more than four digits. You must use a non-digit character or template after YYYY, otherwise the year is always interpreted as four digits. For example, given the following arguments, TO_DATE interprets the five-digit year 20000 as a four-digit year:

```
=> SELECT TO_DATE('200001131', 'YYYYMMDD');
       TO_DATE
-----
2000-01-13
(1 row)
```

Instead, use a non-digit separator after the year. For example:

```
=> SELECT TO_DATE('20000-1131', 'YYYY-MMDD');
       TO_DATE
-----
20000-12-01
(1 row)
```

- In conversions from string to `TIMESTAMP` or `DATE`, the `CC` field is ignored if there is a `YYY`, `YYYY` or `Y,YYY` field. If `CC` is used with `YY` or `Y`, then the year is computed as $(CC-1)*100+YY$.

Examples

```
=> SELECT TO_TIMESTAMP('13 Feb 2009', 'DD Mon YYYY');
       TO_TIMESTAMP
-----
1200-02-13 00:00:00
(1 row)

=> SELECT TO_TIMESTAMP(200120400);
       TO_TIMESTAMP
-----
1976-05-05 01:00:00
(1 row)
```

See Also

[Date/Time Functions](#)

TO_TIMESTAMP_TZ

Converts a string value or a UNIX/POSIX epoch value to a `TIMESTAMP WITH TIME ZONE` type.

Behavior Type

Immutable if single argument form, Stable otherwise.

Syntax

`TO_TIMESTAMP_TZ (expression, pattern)TO_TIMESTAMP (unix-epoch)`

Parameters

<i>expression</i>	Specifies the string value to convert, either CHAR or VARCHAR.
<i>pattern</i>	A CHAR or VARCHAR that specifies an output pattern string. See: <ul style="list-style-type: none"> Template Patterns for Date/Time Formatting Template Patterns for Numeric Formatting
<i>unix-epoch</i>	A DOUBLE PRECISION value that specifies some number of seconds elapsed since midnight UTC of January 1, 1970, not counting leap seconds. INTEGER values are implicitly cast to DOUBLE PRECISION.

Notes

- Millisecond (MS) and microsecond (US) values in a conversion from string to TIMESTAMP are used as part of the seconds after the decimal point. For example `TO_TIMESTAMP ('12:3', 'SS:MS')` is not 3 milliseconds, but 300, because the conversion counts it as 12 + 0.3 seconds. This means for the format SS:MS, the input values 12:3, 12:30, and 12:300 specify the same number of milliseconds. To get three milliseconds, use 12:003, which the conversion counts as $12 + 0.003 = 12.003$ seconds.

Here is a more complex example: `TO_TIMESTAMP ('15:12:02.020.001230', 'HH:MI:SS.MS.US')` is 15 hours, 12 minutes, and 2 seconds + 20 milliseconds + 1230 microseconds = 2.021230 seconds.

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: `'\\"YYYY Month\\"'`
- `TO_TIMESTAMP`, `TO_TIMESTAMP_TZ`, and `TO_DATE` skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:

- `TO_TIMESTAMP('2000 JUN', 'YYYY MON')` is correct.
- `TO_TIMESTAMP('2000 JUN', 'FXYYYY MON')` returns an error, because `TO_TIMESTAMP` expects one space only.
- The `YYYY` conversion from string to `TIMESTAMP` or `DATE` has a restriction if you use a year with more than four digits. You must use a non-digit character or template after `YYYY`, otherwise the year is always interpreted as four digits. For example, given the following arguments, `TO_DATE` interprets the five-digit year 20000 as a four-digit year:

```
=> SELECT TO_DATE('200001131', 'YYYYMMDD');
      TO_DATE
-----
2000-01-13
(1 row)
```

Instead, use a non-digit separator after the year. For example:

```
=> SELECT TO_DATE('20000-1131', 'YYYY-MMDD');
      TO_DATE
-----
20000-12-01
(1 row)
```

- In conversions from string to `TIMESTAMP` or `DATE`, the `CC` field is ignored if there is a `YYY`, `YYYY` or `Y,YYY` field. If `CC` is used with `YY` or `Y`, then the year is computed as $(CC-1)*100+YY$.

Examples

```
=> SELECT TO_TIMESTAMP_TZ('13 Feb 2009', 'DD Mon YYYY');
      TO_TIMESTAMP_TZ
-----
1200-02-13 00:00:00-05
(1 row)

=> SELECT TO_TIMESTAMP_TZ(200120400);
      TO_TIMESTAMP_TZ
-----
1976-05-05 01:00:00-04
(1 row)
```

See Also

[Date/Time Functions](#)

TO_NUMBER

Converts a string value to DOUBLE PRECISION.

Behavior Type

Stable

Syntax

```
TO_NUMBER ( expression, [ pattern ] )
```

Parameters

<i>expression</i>	Specifies the string value to convert, either CHAR or VARCHAR.
<i>pattern</i>	A string value, either CHAR or VARCHAR, that specifies an output pattern string using one of the supported Template Patterns for Numeric Formatting . If you omit this parameter, TO_NUMBER returns a floating point.

Notes

To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: '\\\"YYYY Month\\\"'

Note: To convert a date string to a numeric value, use the appropriate [date/time function](#), such as [EXTRACT](#).

Examples

```
=> SELECT TO_NUMBER('MCL', 'rn');  
TO_NUMBER  
-----
```

```
1950  
(1 row)
```

It the pattern parameter is omitted, the function returns a floating point. For example:

```
=> SELECT TO_NUMBER('-123.456e-01');  
TO_NUMBER  
-----  
-12.3456
```

Template Patterns for Date/Time Formatting

In an output template string (for TO_CHAR), there are certain patterns that are recognized and replaced with appropriately-formatted data from the value to be formatted. Any text that is not a template pattern is copied verbatim. Similarly, in an input template string (for anything other than TO_CHAR), template patterns identify the parts of the input data string to be looked at and the values to be found there.

Note: Vertica uses the ISO 8601:2004 style for date/time fields in Vertica *.log files. For example,
2008-09-16 14:40:59.123 TM Moveout:0x2aaaac002180 [Txn] <INFO>

Certain modifiers can be applied to any template pattern to alter its behavior as described in [Template Pattern Modifiers for Date/Time Formatting](#).

Pattern	Description
HH	Hour of day (00-23)
HH12	Hour of day (01-12)
HH24	Hour of day (00-23)
MI	Minute (00-59)
SS	Second (00-59)
MS	Millisecond (000-999)
US	Microsecond (000000-999999)
SSSS	Seconds past midnight (0-86399)
AM or A.M. or PM or P.M.	Meridian indicator (uppercase)
am or a.m. or pm or p.m.	Meridian indicator (lowercase)
Y,YYY	Year (4 and more digits) with comma
YYYY	Year (4 and more digits)
YYY	Last 3 digits of year

Pattern	Description
YY	Last 2 digits of year
Y	Last digit of year
IYYY	ISO year (4 and more digits)
IYY	Last 3 digits of ISO year
IY	Last 2 digits of ISO year
I	Last digits of ISO year
BC or B.C. or AD or A.D.	Era indicator (uppercase)
bc or b.c. or ad or a.d.	Era indicator (lowercase)
MONTH	Full uppercase month name (blank-padded to 9 chars)
Month	Full mixed-case month name (blank-padded to 9 chars)
month	Full lowercase month name (blank-padded to 9 chars)
MON	Abbreviated uppercase month name (3 chars)
Mon	Abbreviated mixed-case month name (3 chars)
mon	Abbreviated lowercase month name (3 chars)
MM	Month number (01-12)
DAY	Full uppercase day name (blank-padded to 9 chars)
Day	Full mixed-case day name (blank-padded to 9 chars)
day	full lowercase day name (blank-padded to 9 chars)
DY	Abbreviated uppercase day name (3 chars)
Dy	Abbreviated mixed-case day name (3 chars)
dy	Abbreviated lowercase day name (3 chars)
DDD	Day of year (001-366)
DD	Day of month (01-31) for TIMESTAMP

Pattern	Description
	Note: For INTERVAL, DD is day of year (001-366) because day of month is undefined.
D	Day of week (1-7; Sunday is 1)
W	Week of month (1-5) (The first week starts on the first day of the month.)
WW	Week number of year (1-53) (The first week starts on the first day of the year.)
IW	ISO week number of year (The first Thursday of the new year is in week 1.)
CC	Century (2 digits)
J	Julian Day (days since January 1, 4712 BC)
Q	Quarter
RM	Month in Roman numerals (I-XII; I=January) (uppercase)
rm	Month in Roman numerals (i-xii; i=January) (lowercase)
TZ	Time-zone name (uppercase)
tz	Time-zone name (lowercase)

Examples

Use `TO_TIMESTAMP` to convert an expression using the pattern 'YYY MON':

```
=> SELECT TO_TIMESTAMP('2017 JUN', 'YYYY MON');
       TO_TIMESTAMP
-----
2017-06-01 00:00:00
(1 row)
```

Use `TO_DATE` to convert an expression using the pattern 'YYY-MMDD':

```
=> SELECT TO_DATE('2017-1231', 'YYYY-MMDD');
       TO_DATE
-----
2017-12-31
```

(1 row)

Template Pattern Modifiers for Date/Time Formatting

Certain modifiers can be applied to any template pattern to alter its behavior. For example, FMMonth is the Month pattern with the FM modifier.

Modifier	Description
AM	Time is before 12:00
AT	Ignored
JULIAN, JD, J	Next field is Julian Day
FM prefix	Fill mode (suppress padding blanks and zeros) For example: FMMonth Note: The FM modifier suppresses leading zeros and trailing blanks that would otherwise be added to make the output of a pattern fixed width.
FX prefix	Fixed format global option For example: FX Month DD Day
ON	Ignored
PM	Time is on or after 12:00
T	Next field is time
TH suffix	Uppercase ordinal number suffix For example: DDTH
th suffix	Lowercase ordinal number suffix For example: DDth
TM prefix	Translation mode (print localized day and month names based on lc_ messages). For example: TMMonth

Template Patterns for Numeric Formatting

Pattern	Description
9	Value with the specified number of digits
0	Value with leading zeros
. (period)	Decimal point
, (comma)	Group (thousand) separator
PR	Negative value in angle brackets
S	Sign anchored to number (uses locale)
L	Currency symbol (uses locale)
D	Decimal point (uses locale)
G	Group separator (uses locale)
MI	Minus sign in specified position (if number < 0)
PL	Plus sign in specified position (if number > 0)
SG	Plus/minus sign in specified position
RN	Roman numeral (input between 1 and 3999)
TH or th	Ordinal number suffix
V	Shift specified number of digits (see notes)
EEEE	Scientific notation (not implemented yet)

Usage

- A sign formatted using SG, PL, or MI is not anchored to the number; for example:
 - `TO_CHAR(-12, 'S9999')` produces ' -12'
 - `TO_CHAR(-12, 'MI9999')` produces '- 12'
- 9 results in a value with the same number of digits as there are 9s. If a digit is not available it outputs a space.
- TH does not convert values less than zero and does not convert fractional numbers.
- V effectively multiplies the input values by 10^n , where n is the number of digits following V. `TO_CHAR` does not support the use of V combined with a decimal point. For example: 99.9V99 is not allowed.

Geospatial Functions

The following topics describe the Vertica geospatial functions.

Function-Naming Conventions

The geospatial functions use the following naming conventions:

- The `ST_<function_name>` functions are compliant with the latest Open Geospatial Consortium standard OGC SFA-SQL version 1.2.1 (reference. number is OGC 06-104r4, date: 2010-08-04). Currently, some `ST_<function_name>` functions may not support all data types. Each function page contains details about the supported data types.

Note: Some functions, such as `ST_GeomFromText`, are based on previous versions of the standard.

- The `STV_<function_name>` functions are unique to Vertica and not compliant with OGC standards. Each function page explains its functionality in detail.

Verifying Spatial Objects Validity

Many spatial functions do not verify the validity of the parameters. If you pass an invalid spatial object to an ST_ or STV_ function, the function may return an error or produce incorrect results.

To avoid this issue, Micro Focus recommends that you first run ST_IsValid on all spatial objects to verify their validity. If your object is not valid, run STV_IsValidReason to get information about the location of the invalidity.

Note: If you pass a valid polygon to STV_IsValidReason, it returns NULL.

ST_AsText

Creates the Well-Known Text (WKT) representation of a spatial object. Use this function when you need to specify a spatial object in ASCII form.

The [Open Geospatial Consortium \(OGC\)](#) defines the format of a WKT string in the [Simple Feature Access Part 1 - Common Architecture](#) specification.

Behavior Type

Immutable

Syntax

```
ST_AsText( g )
```

Arguments

<i>g</i>	Spatial object for which you want the WKT string, type GEOMETRY or GEOGRAPHY
----------	--

Returns

LONG VARCHAR

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	Yes	No	No

Example

The following example shows how to use ST_AsText.

Retrieve WKB and WKT representations:

```
=> CREATE TABLE locations (id INTEGER, name VARCHAR(100), geom1 GEOMETRY(800),
    geom2 GEOGRAPHY);
CREATE TABLE
=> COPY locations
    (id, geom1x FILLER LONG VARCHAR(800), geom1 AS ST_GeomFromText(geom1x), geom2x FILLER LONG
    VARCHAR (800),
    geom2 AS ST_GeographyFromText(geom2x))
    FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POINT(2 3)|
>> 2|LINESTRING(2 4,1 5)|
>> 3||POLYGON((-70.96 43.27,-70.67 42.95,-66.90 44.74,-67.81 46.08,-67.81 47.20,-69.22 47.43,-71.09
45.25,-70.96 43.27))
>> \.
=> SELECT id, ST_AsText(geom1),ST_AsText(geom2) FROM locations ORDER BY id ASC;
id |          ST_AsText          |          ST_AsText
-----+-----+-----
1 | POINT (2 3)                |
2 | LINESTRING (2 4, 1 5)      |
3 | POLYGON ((-70.96 43.27, -70.67 42.95, -66.9 44.74, -67.81 46.08, -67.81
47.2, -69.22 47.43, -71.09 45.25, -70.96 43.27))
```

(3 rows)

Calculate the length of a WKT using the Vertica SQL function [LENGTH](#):

```
=> SELECT LENGTH(ST_AsText(St_GeomFromText('POLYGON ((-1 2, 0 3, 1 2,
                                           0 1, -1 2))')));

LENGTH
-----
      37
(1 row)
```

See Also

- [ST_AsBinary](#)

ST_Area

Calculates the area of a spatial object.

The units are:

- GEOMETRY objects: spatial reference system identifier (SRID) units
- GEOGRAPHY objects: square meters

Behavior Type

Immutable

Syntax

```
ST_Area( g )
```

Arguments

<i>g</i>	Spatial object for which you want to calculate the area, type GEOMETRY or GEOGRAPHY
----------	---

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Examples

The following examples show how to use ST_Area.

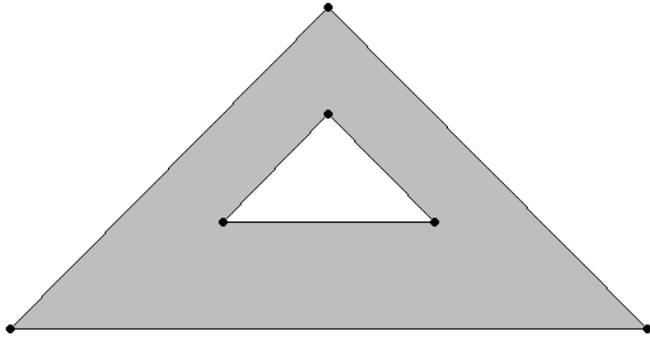
Calculate the area of a polygon:

```
=> SELECT ST_Area(ST_GeomFromText('POLYGON((0 0,1 0,1 1,0 1,0 0))'));
   ST_Area
-----
         1
(1 row)
```

Calculate the area of a multipolygon:

```
=> SELECT ST_Area(ST_GeomFromText('MultiPolygon(((0 0,1 0,1 1,0 1,0 0)),
((2 2,2 3,4 6,3 3,2 2)))'));
   ST_Area
-----
         3
(1 row)
```

Suppose the polygon has a hole, as in the following figure.



Calculate the area, excluding the area of the hole:

```
=> SELECT ST_Area(ST_GeomFromText('POLYGON((2 2,5 5,8 2,2 2),
(4 3,5 4,6 3,4 3))'));
ST_Area
-----
      8
(1 row)
```

Calculate the area of a geometry collection:

```
=> SELECT ST_Area(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((20.5 20.45,
20.51 20.52,20.69 20.32,20.5 20.45)),POLYGON((10 20,30 40,25 50,10 20))'));
ST_Area
-----
150.0073
(1 row)
```

Calculate the area of a geography object:

```
=> SELECT ST_Area(ST_GeographyFromText('POLYGON((20.5 20.45,20.51 20.52,
20.69 20.32,20.5 20.45))'));
ST_Area
-----
84627437.116037
(1 row)
```

ST_AsBinary

Creates the Well-Known Binary (WKB) representation of a spatial object. Use this function when you need to convert an object to binary form for porting spatial data to or from other applications.

The [Open Geospatial Consortium \(OGC\)](#) defines the format of a WKB representation in the [Simple Feature Access Part 1 - Common Architecture](#) specification.

Behavior Type

Immutable

Syntax

```
ST_AsBinary( g )
```

Arguments

<i>g</i>	Spatial object for which you want the WKB, type GEOMETRY or GEOGRAPHY
----------	---

Returns

LONG VARBINARY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	Yes	No	No

Example

The following example shows how to use `ST_AsBinary`.

Retrieve WKB and WKT representations:

```
=> CREATE TABLE locations (id INTEGER, name VARCHAR(100), geom1 GEOMETRY(800), geom2 GEOGRAPHY);
CREATE TABLE
=> COPY locations
  (id, geom1x FILLER LONG VARCHAR(800), geom1 AS ST_GeomFromText(geom1x), geom2x FILLER LONG
  VARCHAR (800),
  geom2 AS ST_GeographyFromText(geom2x))
  FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POINT(2 3)|
>> 2|LINESTRING(2 4,1 5)|
>> 3||POLYGON((-70.96 43.27,-70.67 42.95,-66.90 44.74,-67.81 46.08,-67.81 47.20,-69.22 47.43,-71.09
45.25,-70.96 43.27))
>> \.
=> SELECT id, ST_AsText(geom1),ST_AsText(geom2) FROM locations ORDER BY id ASC;
 id |          ST_AsText          |          ST_AsText
-----+-----+-----
  1 | POINT (2 3)                 |
  2 | LINESTRING (2 4, 1 5)       |
  3 |                             | POLYGON ((-70.96 43.27, -70.67 42.95, -66.9 44.74, -67.81 46.08, -67.81
47.2, -69.22 47.43, -71.09 45.25, -70.96 43.27))
=> SELECT id, ST_AsBinary(geom1),ST_AsBinary(geom2) FROM locations ORDER BY id ASC;
.
.
.
(3 rows)
```

Calculate the length of a WKB using the Vertica SQL function [LENGTH](#):

```
=> SELECT LENGTH(ST_AsBinary(ST_GeomFromText('POLYGON ((-1 2, 0 3, 1 2,
                                0 1, -1 2)))));
 LENGTH
-----
      93
(1 row)
```

See Also

[ST_AsText](#)

ST_Boundary

Calculates the boundary of the specified GEOMETRY object. An object's boundary is the set of points that define the limit of the object.

For a linestring, the boundary is the start and end points. For a polygon, the boundary is a linestring that begins and ends at the same point.

Behavior Type

Immutable

Syntax

```
ST_Boundary( g )
```

Arguments

<i>g</i>	Spatial object for which you want the boundary, type GEOMETRY
----------	---

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes

Polygon	Yes
Multipolygon	Yes
GeometryCollection	No

Examples

The following examples show how to use `ST_Boundary`.

Returns a linestring that represents the boundary:

```
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((-1 -1,2 2,
  0 1,-1 -1)'))));
   ST_AsText
-----
LINESTRING(-1 -1, 2 2, 0 1, -1 -1)
(1 row)
```

Returns a multilinestring that contains the boundaries of both polygons:

```
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((2 2,5 5,8 2,2 2),
  (4 3,5 4,6 3,4 3)'))));
   ST_AsText
-----
MULTILINESTRING ((2 2, 5 5, 8 2, 2 2), (4 3, 5 4, 6 3, 4 3))
(1 row)
```

The boundary of a linestring is its start and end points:

```
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText(
  'LINESTRING(1 1,2 2,3 3,4 4)'));
   ST_AsText
-----
MULTIPOINT (1 1, 4 4)
(1 row)
```

A closed linestring has no boundary because it has no start and end points:

```
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText(
  'LINESTRING(1 1,2 2,3 3,4 4,1 1)'));
   ST_AsText
-----
MULTIPOINT EMPTY
(1 row)
```

ST_Buffer

Creates a GEOMETRY object greater than or equal to a specified distance from the boundary of a spatial object. The distance is measured in Cartesian coordinate units. ST_Buffer does not accept a distance size greater than +1e15 or less than -1e15.

Behavior Type

Immutable

Syntax

```
ST_Buffer( g, d )
```

Arguments

<i>g</i>	Spatial object for which you want to calculate the buffer, type GEOMETRY
<i>d</i>	Distance from the object in Cartesian coordinate units, type FLOAT

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes

Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Usage Tips

- If you specify a positive distance, `ST_Buffer` returns a polygon that represents the points within or equal to the distance outside the object. If you specify a negative distance, `ST_Buffer` returns a polygon that represents the points within or equal to the distance inside the object.
- For points, multipoints, linestrings, and multilinestrings, if you specify a negative distance, `ST_Buffer` returns an empty polygon.
- The Vertica Place version of `ST_Buffer` returns the buffer as a polygon, so the buffer object has corners at its vertices. It does not contain rounded corners.

Example

The following example shows how to use `ST_Buffer`.

Returns a `GEOMETRY` object:

```
=> SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POLYGON((0 1,1 4,4 3,0 1))'),1));
           ST_AsText
-----
POLYGON ((-0.188847498856 -0.159920845081, -1.12155598386 0.649012935089, 0.290814745534
4.76344136152,
0.814758063466 5.02541302048, 4.95372324225 3.68665254814, 5.04124517538 2.45512549204, -
0.188847498856 -0.159920845081))
(1 row)
```

ST_Centroid

Calculates the geometric center—the centroid—of a spatial object. If points or linestrings or both are present in a geometry with polygons, only the polygons contribute to the calculation of the centroid. Similarly, if points are present with linestrings, the points do not contribute to the calculation of the centroid.

To calculate the centroid of a `GEOGRAPHY` object, see the examples for [STV_Geometry](#) and [STV_Geography](#).

Behavior Type

Immutable

Syntax

```
ST_Centroid( g )
```

Arguments

<i>g</i>	Spatial object for which you want to calculate the centroid, type GEOMETRY
----------	--

Returns

GEOMETRY (POINT only)

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

The following examples show how to use ST_Centroid.

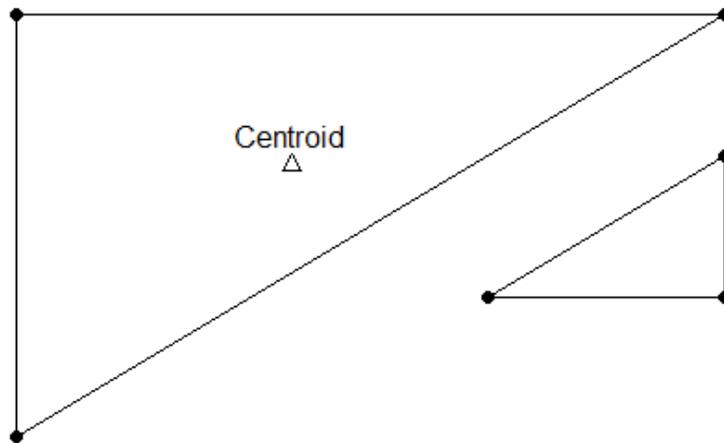
Calculate the centroid for a polygon:

```
=> SELECT ST_AsText(ST_Centroid(ST_GeomFromText('POLYGON((-1 -1,2 2,-1 2,
  -1 -1)'))));
  ST_AsText
-----
POINT (-0 1)
(1 row)
```

Calculate the centroid for a multipolygon:

```
=> SELECT ST_AsText(ST_Centroid(ST_GeomFromText('MULTIPOLYGON(((1 0,2 1,2 0,
  1 0)),((-1 -1,2 2,-1 2,-1 -1)'))));
  ST_AsText
-----
POINT (0.166666666667 0.933333333333)
(1 row)
```

This figure shows the centroid for the multipolygon.



ST_Contains

Determines if a spatial object is entirely inside another spatial object without existing only on its boundary. Both arguments must be the same spatial data type. Either specify two GEOMETRY objects or two GEOGRAPHY objects.

If an object such as a point or linestring only exists along a spatial object's boundary, then `ST_Contains` returns false. The interior of a linestring is all the points on the linestring except the start and end points.

`ST_Contains(g1, g2)` is functionally equivalent to `ST_Within(g2, g1)`.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable

Syntax

```
ST_Contains( g1, g2
            [USING PARAMETERS spheroid={true | false}] )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY or GEOGRAPHY
<i>g2</i>	Spatial object, type GEOMETRY or GEOGRAPHY

Parameters

<code>spheroid = {true false}</code>	(Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False
--	--

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	No	No
Linestring	Yes	Yes	No
Multilinestring	Yes	No	No
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	No
GeometryCollection	Yes	No	No

Compatible GEOGRAPHY pairs:

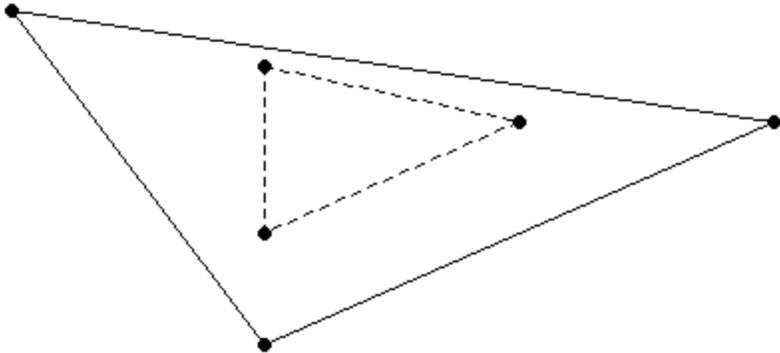
Data Type	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point-Point	Yes	No
Linestring-Point	Yes	No
Polygon-Point	Yes	Yes
Multipolygon-Point	Yes	No

Examples

The following examples show how to use ST_Contains.

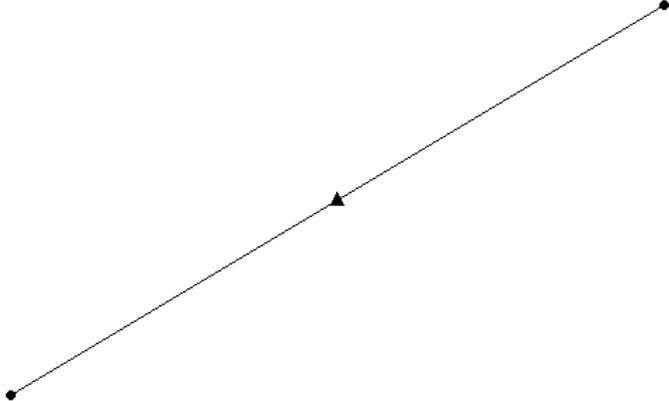
The first polygon does not completely contain the second polygon:

```
=> SELECT ST_Contains(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
  ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
  ST_Contains
  -----
  f
  (1 row)
```



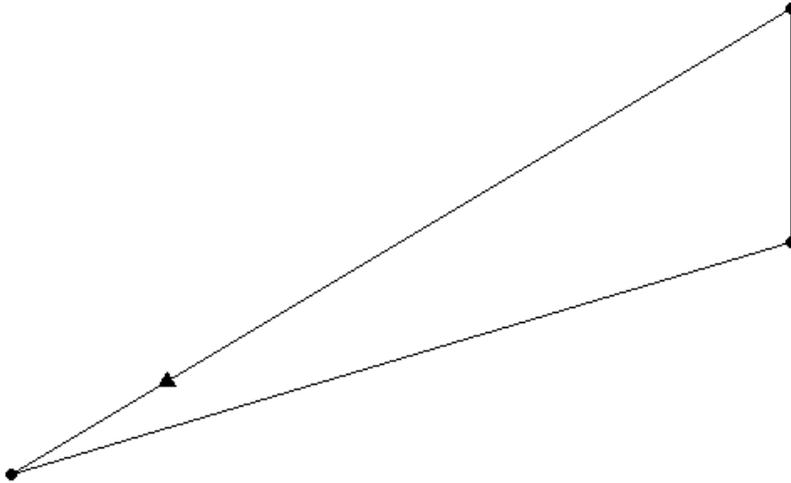
If a point is on a linestring, but not on an end point:

```
=> SELECT ST_Contains(ST_GeomFromText('LINESTRING(20 20,30 30)'),
  ST_GeomFromText('POINT(25 25)'));
  ST_Contains
  -----
  t
(1 row)
```



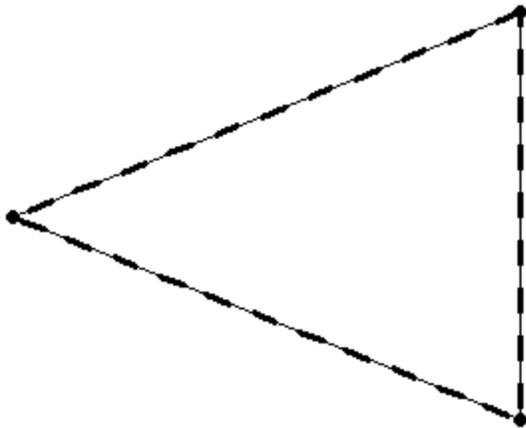
If a point is on the boundary of a polygon:

```
=> SELECT ST_Contains(ST_GeographyFromText('POLYGON((20 20,30 30,30 25,20 20))'),
  ST_GeographyFromText('POINT(20 20)'));
  ST_Contains
  -----
  f
(1 row)
```



Two spatially equivalent polygons:

```
=> SELECT ST_Contains (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),  
    ST_GeomFromText('POLYGON((0 3, -1 2, 0 1, 0 3))'));  
ST_Contains  
-----  
t  
(1 row)
```



See Also

- [ST_Overlaps](#)
- [ST_Within](#)

ST_ConvexHull

Calculates the smallest convex GEOMETRY object that contains a GEOMETRY object.

Behavior Type

Immutable

Syntax

```
ST_ConvexHull( g )
```

Arguments

<i>g</i>	Spatial object for which you want the convex hull, type GEOMETRY
----------	--

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

The following examples show how to use ST_ConvexHull.

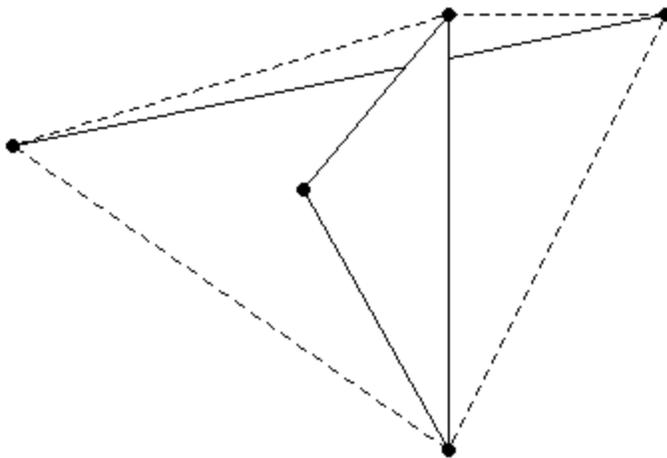
For a pair of points in a geometry collection:

```
=> SELECT ST_AsText(ST_ConvexHull(ST_GeomFromText('GEOMETRYCOLLECTION(  
  POINT(1 1),POINT(0 0)'))));  
      ST_AsText  
-----  
LINESTRING (1 1, 0 0)  
(1 row)
```

For a geometry collection:

```
=> SELECT ST_AsText(ST_ConvexHull(ST_GeomFromText('GEOMETRYCOLLECTION(  
  LINESTRING(2.5 3,-2 1.5), POLYGON((0 1,1 3,1 -2,0 1)))')));  
      ST_AsText  
-----  
POLYGON ((1 -2, -2 1.5, 1 3, 2.5 3, 1 -2))  
(1 row)
```

The solid lines represent the original geometry collection and the dashed lines represent the convex hull.



ST_Crosses

Determines if one GEOMETRY object spatially crosses another GEOMETRY object. If two objects touch only at a border, ST_Crosses returns FALSE.

Two objects spatially cross when both of the following are true:

- The two objects have some, but not all, interior points in common.
- The dimension of the result of their intersection is less than the maximum dimension of the two objects.

Behavior Type

Immutable

Syntax

```
ST_Crosses( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Returns

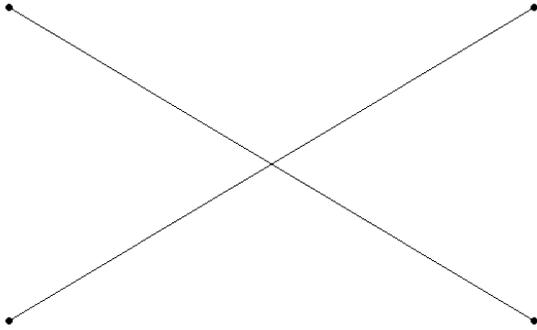
BOOLEAN

Supported Data Types

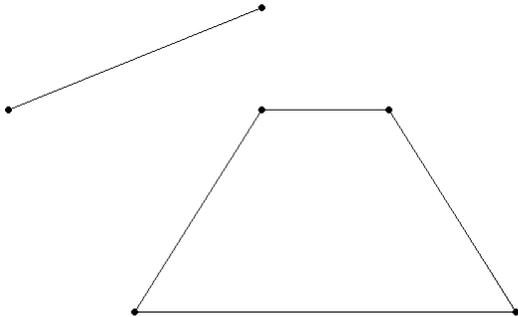
Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

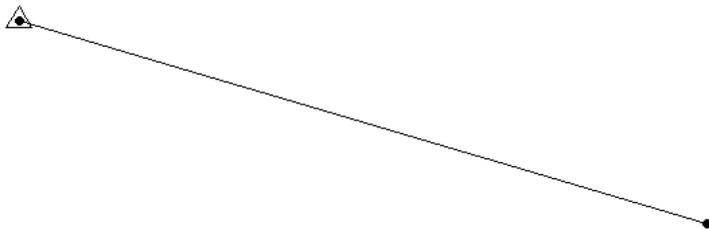
The following examples show how to use `ST_Crosses`.



```
=> SELECT ST_Crosses(ST_GeomFromText('LINESTRING(-1 3,1 4)'),  
  ST_GeomFromText('LINESTRING(-1 4,1 3)'));  
  ST_Crosses  
-----  
t  
(1 row)
```



```
=> SELECT ST_Crosses(ST_GeomFromText('LINESTRING(-1 1,1 2)'),  
  ST_GeomFromText('POLYGON((1 1,0 -1,3 -1,2 1,1 1))'));  
  ST_Crosses  
-----  
f  
(1 row)
```



```
=> SELECT ST_Crosses(ST_GeomFromText('POINT(-1 4)'),
  ST_GeomFromText('LINESTRING(-1 4,1 3)'));
ST_ Crosses
-----
f
(1 row)
```

ST_Difference

Calculates the part of a spatial object that does not intersect with another spatial object.

Behavior Type

Immutable

Syntax

```
ST_Difference( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes

Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

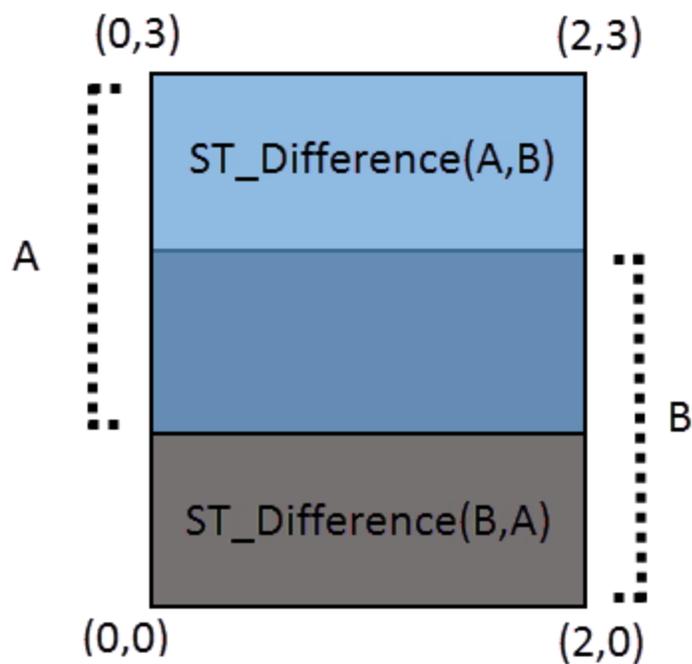
Examples

The following examples show how to use ST_Difference.

Two overlapping linestrings:

```
=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('LINESTRING(0 0,0 2)'),
  ST_GeomFromText('LINESTRING(0 1,0 2)')));
  ST_AsText
-----
LINESTRING (0 0, 0 1)
(1 row)
=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('LINESTRING(0 0,0 3)'),
  ST_GeomFromText('LINESTRING(0 1,0 2)')));
  ST_AsText
-----
MULTILINESTRING ((0 0, 0 1), (0 2, 0 3))
(1 row)
```

Two overlapping polygons:



```
=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('POLYGON((0 1,0 3,2 3,2 1,0 1))'),
  ST_GeomFromText('POLYGON((0 0,0 2,2 2,2 0,0 0))'));
      ST_AsText
-----
POLYGON ((0 2, 0 3, 2 3, 2 2, 0 2))
(1 row)
```

Two non-intersecting polygons:

```
=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('POLYGON((1 1,1 3,3 3,3 1,
  1 1))'),ST_GeomFromText('POLYGON((1 5,1 7,-1 7,-1 5,1 5))'));
      ST_AsText
-----
POLYGON ((1 1, 1 3, 3 3, 3 1, 1 1))
(1 row)
```

ST_Disjoint

Determines if two GEOMETRY objects do not intersect or touch.

If ST_Disjoint returns TRUE for a pair of GEOMETRY objects, ST_Intersects returns FALSE for the same two objects.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable

Syntax

```
ST_Disjoint( g1, g2
             [USING PARAMETERS spheroid={true | false}] )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Parameters

spheroid = {true false}	(Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False
---------------------------	--

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (WGS84)
Point	Yes	Yes
Multipoint	Yes	No
Linestring	Yes	No
Multilinestring	Yes	No
Polygon	Yes	Yes
Multipolygon	Yes	No
GeometryCollection	Yes	No

Compatible GEOGRAPHY pairs:

Data Type	GEOGRAPHY (WGS84)
Point-Point	No
Linestring-Point	No
Polygon-Point	Yes
Multipolygon-Point	No

Examples

The following examples show how to use `ST_Disjoint`.

Two non-intersecting or touching polygons:

```
=> SELECT ST_Disjoint (ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
  ST_GeomFromText('POLYGON((1 0, 1 1, 2 2, 1 0))'));
  ST_Disjoint
-----
t
(1 row)
```

Two intersecting linestrings:

```
=> SELECT ST_Disjoint(ST_GeomFromText('LINESTRING(-1 2,0 3)'),
  ST_GeomFromText('LINESTRING(0 2,-1 3)'));
  ST_Disjoint
-----
f
(1 row)
```

Two polygons touching at a single point:

```
=> SELECT ST_Disjoint (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),
  ST_GeomFromText('POLYGON((0 2, 1 1, 1 2, 0 2))'));
  ST_Disjoint
-----
f
(1 row)
```

See Also

- [ST_Intersects](#)

ST_Distance

Calculates the shortest distance between two spatial objects. For `GEOMETRY` objects, the distance is measured in Cartesian coordinate units. For `GEOGRAPHY` objects, the distance is measured in meters.

Parameters `g1` and `g2` must be both `GEOMETRY` objects or both `GEOGRAPHY` objects.

Behavior Type

Immutable

Syntax

```
ST_Distance( g1, g2  
            [USING PARAMETERS spheroid={ true | false } ] )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY or GEOGRAPHY
<i>g2</i>	Spatial object, type GEOMETRY or GEOGRAPHY

Parameters

spheroid = { true false }	(Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False
-----------------------------	--

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes

Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	No
Multipolygon	Yes	Yes	No
GeometryCollection	Yes	No	No

Compatible GEOGRAPHY pairs:

Data Type	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point-Point	Yes	Yes
Linestring-Point	Yes	Yes
Multilinestring- Point	Yes	Yes
Polygon-Point	Yes	No
Multipoint-Point	Yes	Yes
Multipoint- Multilinestring	Yes	No
Multipolygon- Point	Yes	No

Recommendations

Micro Focus recommends pruning invalid data before using ST_Distance. Invalid geography values could return non-guaranteed results.

Examples

The following examples show how to use ST_Distance.

Distance between two polygons:

```
=> SELECT ST_Distance(ST_GeomFromText('POLYGON((-1 -1,2 2,0 1,-1 -1)'),
                        ST_GeomFromText('POLYGON((5 2,7 4,5 5,5 2))'));
   ST_Distance
-----
                3
(1 row)
```

Distance between a point and a linestring in meters:

```
=> SELECT ST_Distance(ST_GeographyFromText('POINT(31.75 31.25)'),
                    ST_GeographyFromText('LINESTRING(32 32,32 35,40.5 35,32 35,32 32)'));
   ST_Distance
-----
86690.3950562969
(1 row)
```

ST_Envelope

Calculates the minimum bounding rectangle that contains the specified GEOMETRY object.

Behavior Type

Immutable

Syntax

```
ST_Envelope( g )
```

Arguments

<i>g</i>	Spatial object for which you want to find the minimum bounding rectangle, type GEOMETRY
----------	---

Returns

GEOMETRY

Supported Data Types

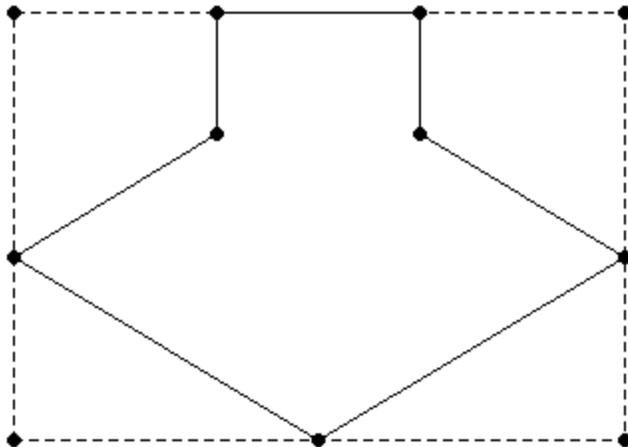
Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Example

The following example shows how to use `ST_Envelope`.

Returns the minimum bounding rectangle:

```
=> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('POLYGON((0 0,1 1,1 2,2 2,
  2 1,3 0,1.5 -1.5,0 0)'))));
      ST_AsText
-----
POLYGON ((0 -1.5, 3 -1.5, 3 2, 0 2, 0 -1.5))
(1 row)
```



ST_Equals

Determines if two spatial objects are spatially equivalent. The coordinates of the two objects and their WKT/WKB representations must match exactly for ST_Equals to return TRUE.

The order of the points do not matter in determining spatial equivalence:

- `LINESTRING(1 2, 4 3)` equals `LINESTRING(4 3, 1 2)`.
- `POLYGON ((0 0, 1 1, 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0))` equals `POLYGON((1 1 , 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0, 1 1))`.
- `MULTILINESTRING((1 2, 4 3),(0 0, -1 -4))` equals `MULTILINESTRING((0 0, -1 -4),(1 2, 4 3))`.

Coordinates are stored as FLOAT types. Thus, rounding errors are expected when importing Well-Known Text (WKT) values because the limitations of floating-point number representation.

g1 and *g2* must both be GEOMETRY objects or both be GEOGRAPHY objects. Also, *g1* and *g2* cannot both be of type GeometryCollection.

Behavior Type

Immutable

Syntax

```
ST_Equals( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object to compare to <i>g2</i> , type GEOMETRY or GEOGRAPHY
<i>g2</i>	Spatial object to compare to <i>g1</i> , type GEOMETRY or GEOGRAPHY

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how to use ST_Equals.

Two linestrings:

```
=> SELECT ST_Equals (ST_GeomFromText('LINESTRING(-1 2, 0 3)'),
  ST_GeomFromText('LINESTRING(0 3, -1 2)'));
  ST_Equals
  -----
  t
(1 row)
```

Two polygons:

```
=> SELECT ST_Equals (ST_GeographyFromText('POLYGON((43.22 42.21,40.3 39.88,
  42.1 50.03,43.22 42.21))'),ST_GeographyFromText('POLYGON((43.22 42.21,
  40.3 39.88,42.1 50.31,43.22 42.21))'));
  ST_Equals
  -----
  f
(1 row)
```

ST_GeographyFromText

Converts a Well-Known Text (WKT) string into its corresponding GEOGRAPHY object. Use this function to convert a WKT string into the format expected by the Vertica Place functions.

A GEOGRAPHY object is a spatial object with coordinates (longitude, latitude) defined on the surface of the earth. Coordinates are expressed in degrees (longitude, latitude) from reference planes dividing the earth.

The maximum size of a GEOGRAPHY object is 10 MB. If you pass a WKT to ST_GeographyFromText, the result is a spatial object whose size is greater than 10 MB, ST_GeographyFromText returns an error.

The [Open Geospatial Consortium \(OGC\)](#) defines the format of a WKT string in Section 7 in the [Simple Feature Access Part 1 - Common Architecture](#) specification.

Behavior Type

Immutable

Syntax

```
ST_GeographyFromText( wkt [ USING PARAMETERS ignore_errors={'y' | 'n'} ] )
```

Arguments

<i>wkt</i>	Well-Known Text (WKT) string of a GEOGRAPHY object, type LONG VARCHAR
<i>ignore_errors</i>	(Optional) ST_GeographyFromText returns the following, based on the parameters supplied: <ul style="list-style-type: none">• NULL—If <i>wkt</i> is invalid and <i>ignore_errors</i>='y'.• Error—If <i>wkt</i> is invalid and <i>ignore_errors</i>='n' or is unspecified.

Returns

GEOGRAPHY

Supported Data Types

Data Type	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	No	No

Example

The following example shows how to use `ST_GeographyFromText`.

Convert WKT into a GEOGRAPHY object:

```
=> CREATE TABLE wkt_ex (g GEOGRAPHY);
CREATE TABLE
=> INSERT INTO wkt_ex VALUES(ST_GeographyFromText('POLYGON((1 2,3 4,2 3,1 2))'));
OUTPUT
-----
      1
(1 row)
```

ST_GeographyFromWKB

Converts a Well-Known Binary (WKB) value into its corresponding GEOGRAPHY object. Use this function to convert a WKB into the format expected by Vertica Place functions.

A GEOGRAPHY object is a spatial object defined on the surface of the earth. Coordinates are expressed in degrees (longitude, latitude) from reference planes dividing the earth. All calculations are in meters.

The maximum size of a GEOGRAPHY object is 10 MB. If you pass a WKB to ST_GeographyFromWKB that results in a spatial object whose size is greater than 10 MB, ST_GeographyFromWKB returns an error.

The [Open Geospatial Consortium \(OGC\)](#) defines the format of a WKB representation in Section 8 in the [Simple Feature Access Part 1 - Common Architecture](#) specification.

Behavior Type

Immutable

Syntax

```
ST_GeographyFromWKB( wkb [ USING PARAMETERS ignore_errors={'y' | 'n'} ] )
```

Arguments

<i>wkb</i>	Well-Known Binary (WKB) value of a GEOGRAPHY object, type LONG VARBINARY
<i>ignore_errors</i>	(Optional) ST_GeographyFromWKB returns the following, based on the parameters supplied: <ul style="list-style-type: none">• NULL—If <i>wkb</i> is invalid and <i>ignore_errors</i>='y'.• Error—If <i>wkb</i> is invalid and <i>ignore_errors</i>='n' or is unspecified.

Returns

GEOGRAPHY

Supported Data Types

Data Type	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes
Multipoint	Yes	Yes

Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	No	No

Example

The following example shows how to use `ST_GeographyFromWKB`.

Convert WKB into a GEOGRAPHY object:

```
=> CREATE TABLE wkb_ex (g GEOGRAPHY);
CREATE TABLE
=> INSERT INTO wkb_ex VALUES(ST_GeographyFromWKB(X'0103000000010000000 ... ));
OUTPUT
-----
      1
(1 row)
```

ST_GeoHash

Returns a GeoHash in the shape of the specified geometry.

Behavior Type

Immutable

Syntax

```
ST_GeoHash( SpatialObject [ USING PARAMETERS numchars=n] )
```

Arguments

<i>Spatial object</i>	A GEOMETRY or GEOGRAPHY spatial object. Inputs must be in polar coordinates (-180 <= x <= 180 and -90 <= y <= 90) for all points inside the given geometry.
-----------------------	---

<i>n</i>	Specifies the length, in characters, of the returned GeoHash.
----------	---

Returns

GEOHASH

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	Yes	No	No

Examples

The following examples show how to use ST_PointFromGeoHash.

Generate a full precision GeoHash for the specified geometry:

```
=> SELECT ST_GeoHash(ST_GeographyFromText('POINT(3.14 -1.34)'));
ST_GeoHash
-----
kpf0rkn3zmcswks75010
(1 row)
```

Generate a GeoHash based on the first five characters of the specified geometry:

```
=> select ST_GeoHash(ST_GeographyFromText('POINT(3.14 -1.34)')USING PARAMETERS numchars=5);
ST_GeoHash
-----
kpf0r
```

(1 row)

ST_GeometryN

Returns the n^{th} geometry within a geometry object.

If n is out of range of the index, then NULL is returned.

Behavior Type

Immutable

Syntax

`ST_GeometryN(g , n)`

Arguments

g	Spatial object of type GEOMETRY.
n	The geometry's index number, 1-based.

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes

Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how to use ST_GeometryN.

Return the second geometry in a multipolygon:

```
=> CREATE TABLE multipolygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multipolygon_geom(gid, gx FILLER LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>9|MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
>>\.
=> SELECT gid, ST_AsText(ST_GeometryN(geom, 2)) FROM multipolygon_geom;
gid |          ST_AsText
-----+-----
  9 | POLYGON ((0 0, 0 5, 1 0, 0 0))
(1 row)
```

Return all the geometries within a multipolygon:

```
=> CREATE TABLE multipolygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multipolygon_geom(gid, gx FILLER LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>9|MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
>>\.
=> CREATE TABLE series_numbers (numbs int);
CREATE TABLE
=> COPY series_numbers FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> 5
>> \.
=> SELECT numbs, ST_AsText(ST_GeometryN(geom, numbs))
FROM multipolygon_geom, series_numbers
```

```
WHERE ST_AsText(ST_GeometryN(geom, numbs)) IS NOT NULL
ORDER BY numbs ASC;
numbs |          ST_AsText
-----+-----
1 | POLYGON ((2 6, 2 9, 6 9, 7 7, 4 6, 2 6))
2 | POLYGON ((0 0, 0 5, 1 0, 0 0))
3 | POLYGON ((0 2, 2 5, 4 5, 0 2))
(3 rows)
```

See Also

[ST_NumGeometries](#)

ST_GeometryType

Determines the class of a spatial object.

Behavior Type

Immutable

Syntax

```
ST_GeometryType( g )
```

Arguments

<i>g</i>	Spatial object for which you want the class, type GEOMETRY or GEOGRAPHY
----------	---

Returns

VARCHAR

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
-----------	----------	----------------------------

Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Example

The following example shows how to use `ST_GeometryType`.

Returns spatial class:

```
=> SELECT ST_GeometryType(ST_GeomFromText('GEOMETRYCOLLECTION(LINESTRING(1 1,
  2 2), POLYGON((1 3,4 5,2 2,1 3))')));
   ST_GeometryType
-----
ST_GeometryCollection
(1 row)
```

ST_GeomFromGeoHash

Returns a polygon in the shape of the specified GeoHash.

Behavior Type

Immutable

Syntax

```
ST_GeomFromGeoHash(GeoHash)
```

Arguments

<i>GeoHash</i>	A valid GeoHash string of arbitrary length.
----------------	---

Returns

GEOGRAPHY

Examples

The following examples show how to use `ST_GeomFromGeoHash`.

Converts a GeoHash string to a Geography object and back to a GeoHash

```
=> SELECT ST_GeoHash(ST_GeomFromGeoHash('vert1c9'));
ST_GeoHash
-----
vert1c9
(1 row)
```

Returns a polygon of the specified GeoHash and uses `ST_AsText` to convert the polygon, rectangle map tile, into Well-Known Text:

```
=> SELECT ST_AsText(ST_GeomFromGeoHash('drt3jj9n4dpcbcdef'));
ST_AsText
-----
POLYGON ((-71.1459699298 42.3945346513, -71.1459699297 42.3945346513, -71.1459699297 42.3945346513, -
71.1459699298 42.3945346513, -71.1459699298 42.3945346513))
(1 row)
```

Returns multiple polygons and their areas for the specified GeoHashes. The polygon for the high level GeoHash (1234) has a significant area, while the low level GeoHash (1234567890bcdefhjkmn) has an area of zero.

```
=> SELECT ST_Area(short) short_area, ST_AsText(short) short_WKT, ST_Area(long) long_area, ST_AsText
(long) long_WKT from (SELECT ST_GeomFromGeoHash('1234') short, ST_GeomFromGeoHash
('1234567890bcdefhjkmn') long) as foo;
-[ RECORD 1 ]-----
short_area | 24609762.8991076
short_WKT  | POLYGON ((-122.34375 -88.2421875, -121.9921875 -88.2421875, -121.9921875 -88.06640625, -
122.34375 -88.06640625, -122.34375 -88.2421875))
long_area  | 0
long_WKT   | POLYGON ((-122.196077187 -88.2297377551, -122.196077187 -88.2297377551, -122.196077187 -
```

```
88.2297377551, -122.196077187 -88.2297377551, -122.196077187 -88.2297377551))
```

ST_GeomFromText

Converts a Well-Known Text (WKT) string into its corresponding GEOMETRY object. Use this function to convert a WKT string into the format expected by the Vertica Place functions.

A GEOMETRY object is a spatial object defined by the coordinates of a plane. Coordinates are expressed as points on a Cartesian plane (x,y). SRID values of 0 to $2^{32}-1$ are valid. SRID values outside of this range will generate an error.

The maximum size of a GEOMETRY object is 10 MB. If you pass a WKT to ST_GeomFromText and the result is a spatial object whose size is greater than 10 MB, ST_GeomFromText returns an error.

The [Open Geospatial Consortium \(OGC\)](#) defines the format of a WKT representation. See section 7 in the [Simple Feature Access Part 1 - Common Architecture](#) specification.

Behavior Type

Immutable

Syntax

```
ST_GeomFromText( wkt [, srid] [ USING PARAMETERS ignore_errors={'y' | 'n'} ] )
```

Arguments

<i>wkt</i>	Well-Known Text (WKT) string of a GEOMETRY object, type LONG VARCHAR.
<i>srid</i>	(Optional when not performing operations) Spatial reference system identifier (SRID) of the GEOMETRY object, type INTEGER. The SRID is stored in the GEOMETRY object, but does not influence the results of spatial computations.
<i>ignore_errors</i>	(Optional) ST_GeomFromText returns the following, based on parameters supplied:

	<ul style="list-style-type: none">• NULL—If <i>wkt</i> is invalid and <i>ignore_errors</i>='y'.• Error—If <i>wkt</i> is invalid and <i>ignore_errors</i>='n' or is unspecified.
--	--

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	No

Example

The following example shows how to use `ST_GeomFromText`.

Convert WKT into a GEOMETRY object:

```
=> SELECT ST_Area(ST_GeomFromText('POLYGON((1 1,2 3,3 5,0 5,1 -2,0 0,1 1))'));
   ST_Area
-----
          6
(1 row)
```

ST_GeomFromWKB

Converts the Well-Known Binary (WKB) value to its corresponding GEOMETRY object. Use this function to convert a WKB into the format expected by many of the Vertica Place functions.

A GEOMETRY object is a spatial object with coordinates (x,y) defined in the Cartesian plane.

The maximum size of a GEOMETRY object is 10 MB. If you pass a WKB to ST_GeomFromWKB and the result is a spatial object whose size is greater than 10 MB, ST_GeomFromWKB returns an error.

The [Open Geospatial Consortium \(OGC\)](#) defines the format of a WKB representation in section 8 in the [Simple Feature Access Part 1 - Common Architecture](#) specification.

Behavior Type

Immutable

Syntax

```
ST_GeomFromWKB( wkb [, srid] [ USING PARAMETERS ignore_errors={'y'|'n'} ] )
```

Arguments

<i>wkb</i>	Well-Known Binary (WKB) value of a GEOMETRY object, type LONG VARBINARY
<i>srid</i>	(Optional) Spatial reference system identifier (SRID) of the GEOMETRY object, type INTEGER. The SRID is stored in the GEOMETRY object, but does not influence the results of spatial computations.
<i>ignore_errors</i>	(Optional) ST_GeomFromWKB returns the following, based on the parameters supplied: <ul style="list-style-type: none">• NULL—If <i>wkb</i> is invalid and <i>ignore_errors</i>='y'.• Error—If <i>wkb</i> is invalid and <i>ignore_errors</i>='n' or is unspecified.

ST_Intersection

Calculates the set of points shared by two GEOMETRY objects.

Behavior Type

Immutable

Syntax

```
ST_Intersection( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Returns

GEOMETRY

Supported Data Types

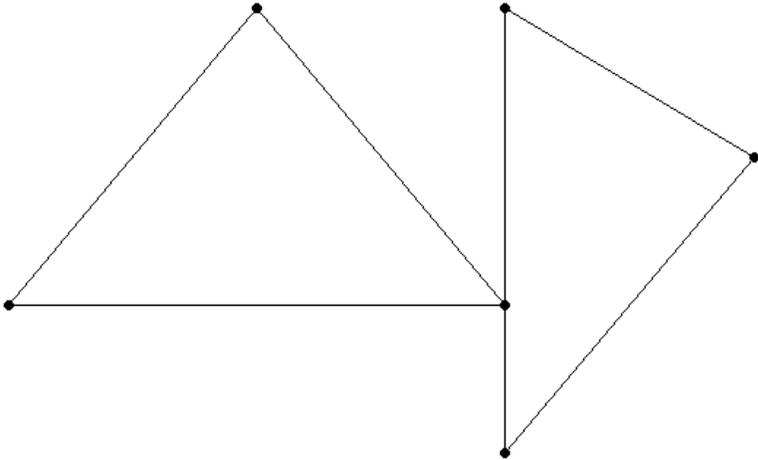
Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes

Multipolygon	Yes
GeometryCollection	Yes

Examples

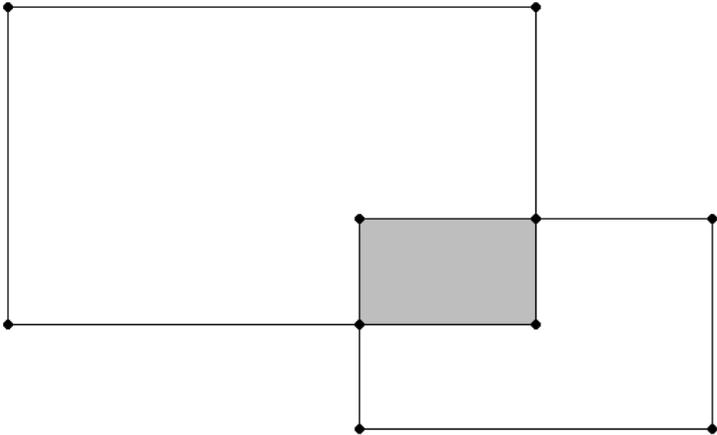
The following examples show how to use ST_Intersection.

Two polygons intersect at a single point:



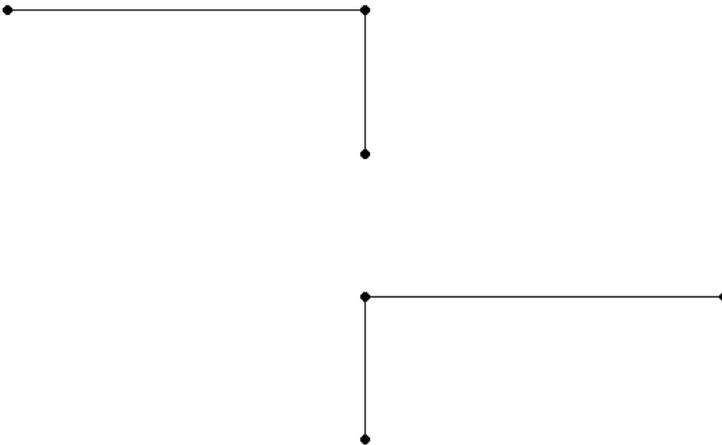
```
=> SELECT ST_AsText(ST_Intersection(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,
  0 2))'),ST_GeomFromText('POLYGON((-1 2,0 0,-2 0,-1 2))'));
  ST_AsText
-----
POINT(0 0)
(1 row)
```

Two polygons:



```
=> SELECT ST_AsText(ST_Intersection(ST_GeomFromText('POLYGON((1 2,1 5,4 5,
  4 2,1 2))'), ST_GeomFromText('POLYGON((3 1,3 3,5 3,5 1,3 1)'))));
  ST_AsText
-----
POLYGON ((4 3, 4 2, 3 2, 3 3, 4 3))
(1 row)
```

Two non-intersecting linestrings:



```
=> SELECT ST_AsText(ST_Intersection(ST_GeomFromText('LINESTRING(1 1,1 3,3 3)'),
  ST_GeomFromText('LINESTRING(1 5,1 7,-1 7)')));
  ST_AsText
-----
GEOMETRYCOLLECTION EMPTY
(1 row)
```

ST_Intersects

Determines if two GEOMETRY or GEOGRAPHY objects intersect or touch at a single point. If ST_Disjoint returns TRUE, ST_Intersects returns FALSE for the same GEOMETRY or GEOGRAPHY objects.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable

Syntax

```
ST_Intersects(g1, g2  
             [USING PARAMETERS bbox={true | false}, spheroid={true | false}])
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Parameters

<code>bbox = {true false}</code>	Boolean. Intersects the bounding box of <i>g1</i> and <i>g2</i> . Default: False
<code>spheroid = {true false}</code>	(Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (WGS84)
Point	Yes	Yes
Multipoint	Yes	No
Linestring	Yes	No
Multilinestring	Yes	No

Polygon	Yes	Yes
Multipolygon	Yes	No
GeometryCollection	Yes	No

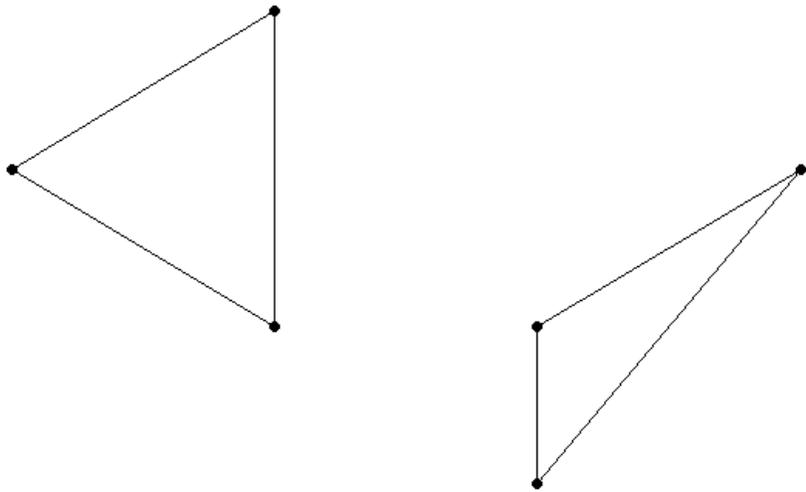
Compatible GEOGRAPHY pairs:

Data Type	GEOGRAPHY (WGS84)
Point-Point	No
Linestring-Point	No
Polygon-Point	Yes
Multipolygon-Point	No

Examples

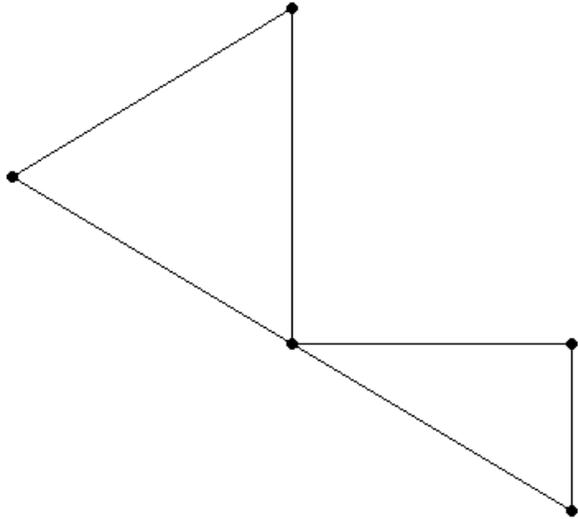
The following examples show how to use ST_Intersects.

Two polygons do not intersect or touch:



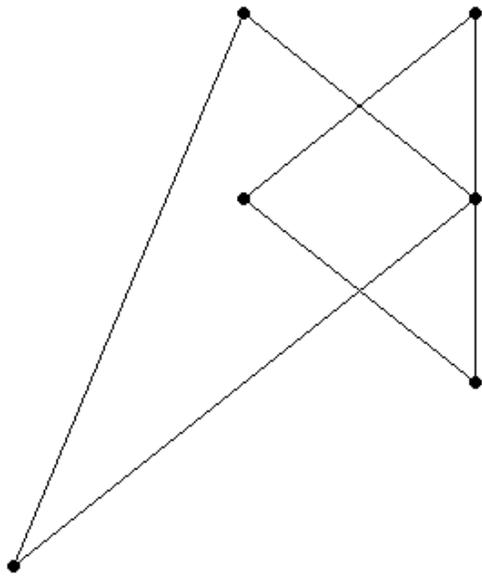
```
=> SELECT ST_Intersects (ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2)'),
    ST_GeomFromText('POLYGON((1 0,1 1,2 2,1 0)'));
    ST_Intersects
    -----
    f
    (1 row)
```

Two polygons touch at a single point:



```
=> SELECT ST_Intersects (ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),  
    ST_GeomFromText('POLYGON((1 0,1 1,0 1,1 0))'));  
ST_Intersects  
-----  
t  
(1 row)
```

Two polygons intersect:



```
=> SELECT ST_Intersects (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),  
    ST_GeomFromText('POLYGON((0 2, -1 3, -2 0, 0 2))'));  
ST_Intersects  
-----  
t  
(1 row)
```

See Also

[ST_Disjoint](#)

ST_IsEmpty

Determines if a spatial object represents the empty set. An empty object has no dimension.

Behavior Type

Immutable

Syntax

```
ST_IsEmpty( g )
```

Arguments

<i>g</i>	Spatial object, type GEOMETRY or GEOGRAPHY
----------	--

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes

Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	Yes	No	No

Example

The following example shows how to use ST_IsEmpty.

An empty polygon:

```
=> SELECT ST_IsEmpty(ST_GeomFromText('GeometryCollection EMPTY'));
   ST_IsEmpty
-----
t
(1 row)
```

ST_IsSimple

Determines if a spatial object does not intersect itself or touch its own boundary at any point.

Behavior Type

Immutable

Syntax

```
ST_IsSimple( g )
```

Arguments

<i>g</i>	Spatial object, type GEOMETRY or GEOGRAPHY
----------	--

Returns

BOOLEAN

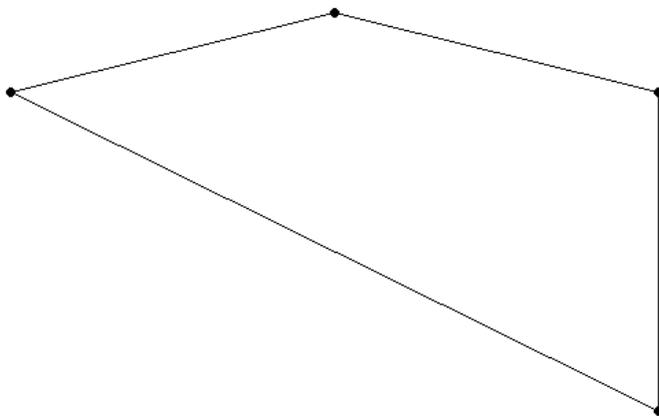
Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	No
Linestring	Yes	Yes
Multilinestring	Yes	No
Polygon	Yes	Yes
Multipolygon	Yes	No
GeometryCollection	No	No

Examples

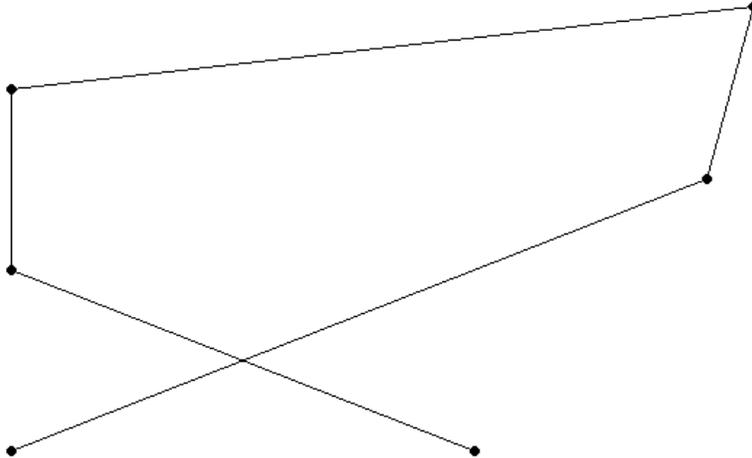
The following examples show how to use `ST_IsSimple`.

Polygon does not intersect itself:



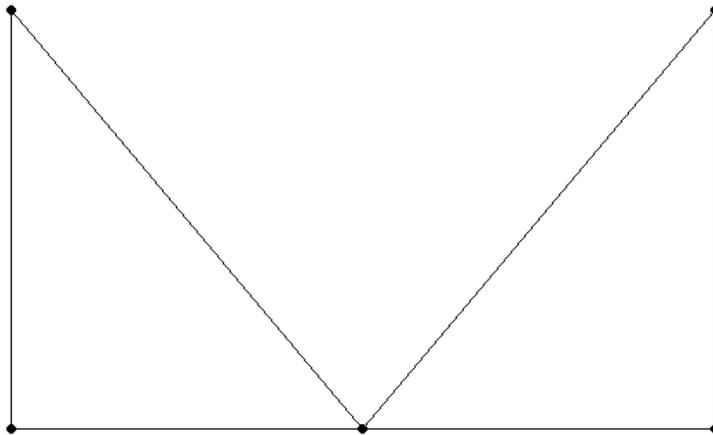
```
=> SELECT ST_IsSimple(ST_GeomFromText('POLYGON((-1 2,0 3,1 2,1 -2,-1 2))'));
ST_IsSimple
-----
t
(1 row)
```

Linestring intersects itself.:



```
=> SELECT ST_IsSimple(ST_GeographyFromText('LINESTRING(10 10,25 25,26 34.5,
      10 30,10 20,20 10)'));
St_IsSimple
-----
f
(1 row)
```

Linestring touches its interior at one or more locations:



```
=> SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(0 0,0 1,1 0,2 1,2 0,0 0)'));
ST_IsSimple
-----
f
(1 row)
```

ST_IsValid

Determines if a spatial object is well formed or valid. If the object is valid, `ST_IsValid` returns TRUE; otherwise, it returns FALSE. Use `STV_IsValidReason` to identify the location of the invalidity.

Spatial validity applies only to polygons and multipolygons. A polygon or multipolygon is valid if all of the following are true:

- The polygon is closed; its start point is the same as its end point.
- Its boundary is a set of linestrings.
- The boundary does not touch or cross itself.
- Any polygons in the interior do not touch the boundary of the exterior polygon except at a vertex.

The [Open Geospatial Consortium \(OGC\)](#) defines the validity of a polygon in section 6.1.11.1 of the [Simple Feature Access Part 1 - Common Architecture](#) specification.

If you are not sure if a polygon is valid, run `ST_IsValid` first. If you pass an invalid spatial object to a Vertica Place function, the function fails or returns incorrect results.

Behavior Type

Immutable

Syntax

```
ST_IsValid( g )
```

Arguments

<i>g</i>	Geospatial object to test for validity, value of type GEOMETRY or GEOGRAPHY (WGS84).
----------	--

Returns

BOOLEAN

Supported Data Types

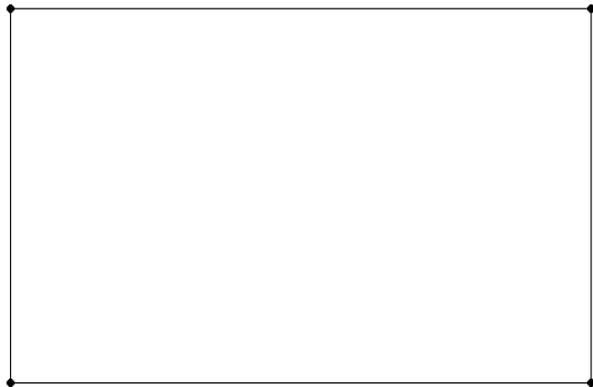
Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
-----------	----------	----------------------------	-------------------

Point	Yes	No	No
Multipoint	Yes	No	No
Linestring	Yes	No	No
Multilinestring	Yes	No	No
Polygon	Yes	No	Yes
Multipolygon	Yes	No	No
GeometryCollection	Yes	No	No

Examples

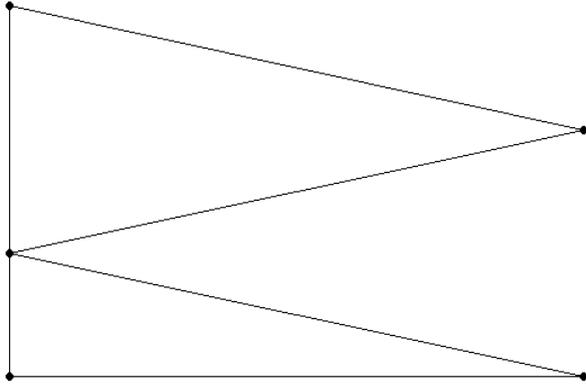
The following examples show how to use ST_IsValid.

Valid polygon:



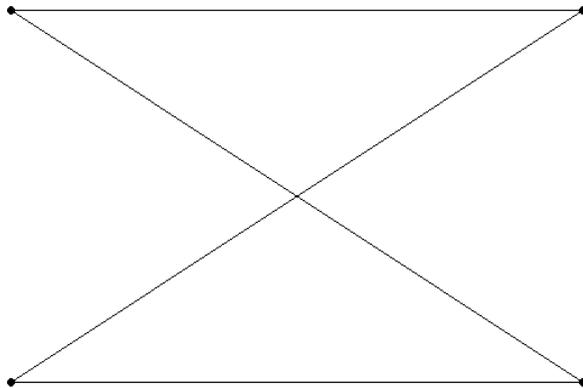
```
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1 1,1 3,3 3,3 1,1 1))'));
   ST_IsValid
-----
t
(1 row)
```

Invalid polygon:



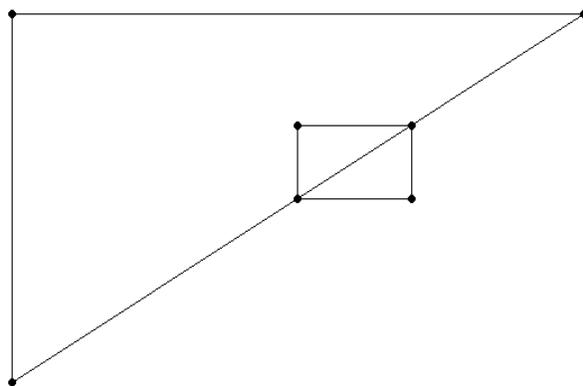
```
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1 3,3 2,1 1,3 0,1 0,1 3))'));
   ST_IsValid
-----
          f
(1 row)
```

Invalid polygon:



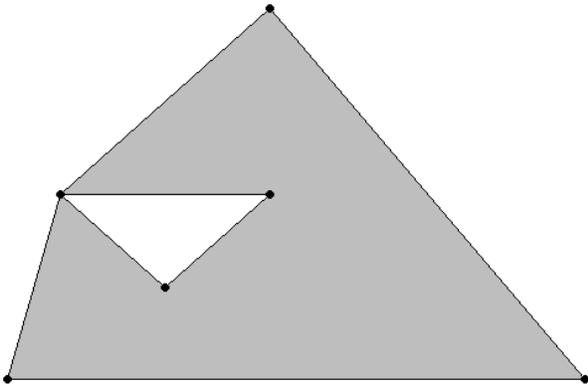
```
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0,2 2,0 2,2 0,0 0))'));
   ST_IsValid
-----
          f
(1 row)
```

Invalid multipolygon:.



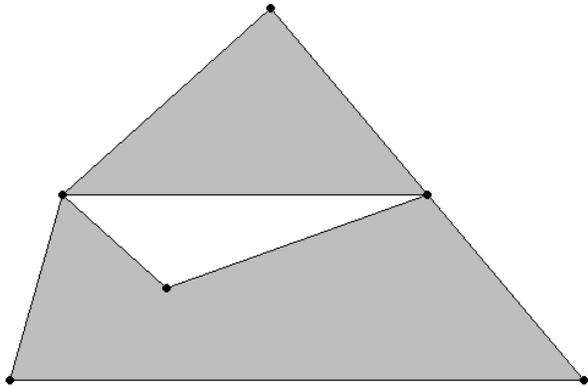
```
=> SELECT ST_IsValid(ST_GeomFromText('MULTIPOLYGON(((0 0, 0 1, 1 1, 0 0)),  
  ((0.5 0.5, 0.7 0.5, 0.7 0.7, 0.5 0.7, 0.5 0.5)))'));  
  ST_IsValid  
-----  
  f  
(1 row)
```

Valid polygon with hole:



```
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1 1,3 3,6 -1,0.5 -1,1 1),  
  (1 1,3 1,2 0,1 1))'));  
  ST_IsValid  
-----  
  t  
(1 row)
```

Invalid polygon with hole:



```
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1 1,3 3,6 -1,0.5 -1,1 1),  
  (1 1,4.5 1,2 0,1 1))'));  
  ST_IsValid  
-----  
  f  
(1 row)
```

ST_Length

Calculates the length of a spatial object. For GEOMETRY objects, the length is measured in Cartesian coordinate units. For GEOGRAPHY objects, the length is measured in meters.

Calculates the length as follows:

- The length of a point or multipoint object is 0.
- The length of a linestring is the sum of the lengths of each line segment. The length of a line segment is the distance from the start point to the end point.
- The length of a polygon is the sum of the lengths of the exterior boundary and any interior boundaries.
- The length of a multilinestring, multipolygon, or geometrycollection is the sum of the lengths of all the objects it contains.

Note: ST_Length does not calculate the length of WKTs or WKBs. To calculate the lengths of those objects, use the Vertica [LENGTH](#) SQL function with ST_AsBinary or ST_AsText.

Behavior Type

Immutable

Syntax

```
ST_Length( g )
```

Arguments

<i>g</i>	Spatial object for which you want to calculate the length, type GEOMETRY or GEOGRAPHY
----------	---

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Examples

The following examples show how to use ST_Length.

Returns length in Cartesian coordinate units:

```
=> SELECT ST_Length(ST_GeomFromText('LINESTRING(-1 -1,2 2,4 5,6 7)'));
      ST_Length
-----
10.6766190873295
(1 row)
```

Returns length in meters:

```
=> SELECT ST_Length(ST_GeographyFromText('LINESTRING(-56.12 38.26,-57.51 39.78,
      -56.37 45.24)'));
      ST_Length
-----
821580.025733461
(1 row)
```

ST_NumGeometries

Returns the number of geometries contained within a spatial object. Single GEOMETRY or GEOGRAPHY objects return 1 and empty objects return NULL.

Behavior Type

Immutable

Syntax

```
ST_NumGeometries( g )
```

Arguments

<i>g</i>	Spatial object of type GEOMETRY or GEOGRAPHY
----------	--

Returns

INTEGER

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following example shows how to use ST_NumGeometries.

Return the number of geometries:

```
=> SELECT ST_NumGeometries(ST_GeomFromText('MULTILINESTRING ((1 5, 2 4, 5 3, 6 6), (3 5, 3 7))'));
   ST_NumGeometries
-----
                2
(1 row)
```

See Also

[ST_GeometryN](#)

ST_NumPoints

Calculates the number of vertices of a spatial object, empty objects return NULL.

The first and last vertex of polygons and multipolygons are counted separately.

Behavior Type

Immutable

Syntax

```
ST_NumPoints( g )
```

Arguments

<i>g</i>	Spatial object for which you want to count the vertices, type GEOMETRY or GEOGRAPHY
----------	---

Returns

INTEGER

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how to use ST_NumPoints.

Returns the number of vertices in a linestring:

```
=> SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(1.33 1.56,2.31 3.4,2.78 5.82,
      3.76 3.9,4.11 3.27,5.85 4.34,6.9 4.231,7.61 5.77)'));
ST_NumPoints
-----
                8
(1 row)
```

Use ST_Boundary and ST_NumPoints to return the number of vertices of a polygon:

```
=> SELECT ST_NumPoints(ST_Boundary(ST_GeomFromText('POLYGON((1 2,1 4,
      2 5,3 6,4 6,5 5,4 4,3 3,1 2))')));
ST_NumPoints
-----
                9
(1 row)
```

ST_Overlaps

Determines if a GEOMETRY object shares space with another GEOMETRY object, but is not completely contained within that object. They must overlap at their interiors. If two objects touch at a single point or intersect only along a boundary, they do not overlap. Both parameters must have the same dimension; otherwise, ST_Overlaps returns FALSE.

Behavior Type

Immutable

Syntax

```
ST_Overlaps ( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Returns

BOOLEAN

Supported Data Types

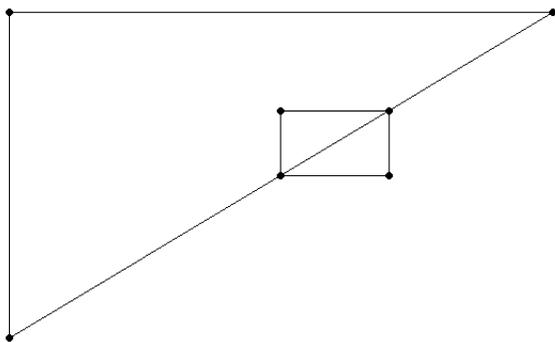
Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes

Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

The following examples show how to use `ST_Overlaps`.

`Polygon_1` overlaps but does not completely contain `Polygon_2`:



```
=> SELECT ST_Overlaps(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 0 0))'),
  ST_GeomFromText('POLYGON((0.5 0.5, 0.7 0.5, 0.7 0.7, 0.5 0.7, 0.5 0.5))'));
 ST_Overlaps
-----
t
(1 row)
```

Two objects with different dimensions:

```
=> SELECT ST_Overlaps(ST_GeomFromText('LINESTRING(2 2,4 4)'),
  ST_GeomFromText('POINT(3 3)'));
 ST_Overlaps
-----
f
(1 row)
```

ST_PointFromGeoHash

Returns the center point of the specified GeoHash.

Behavior Type

Immutable

Syntax

`ST_PointFromGeoHash(GeoHash)`

Arguments

<i>GeoHash</i>	A valid GeoHash string of arbitrary length.
----------------	---

Returns

GEOGRAPHY POINT

Examples

The following examples show how to use `ST_PointFromGeoHash`.

Returns the geography point of a high-level GeoHash and uses [ST_AsText](#) to convert that point into Well-Known Text:

```
=> SELECT ST_AsText(ST_PointFromGeoHash('dr'));
ST_AsText
-----
POINT (-73.125 42.1875)
(1 row)
```

Returns the geography point of a detailed GeoHash and uses `ST_AsText` to convert that point into Well-Known Text:

```
=> SELECT ST_AsText(ST_PointFromGeoHash('1234567890bcdefhjkmn'));
ST_AsText
-----
POINT (-122.196077187 -88.2297377551)
(1 row)
```

ST_PointN

Finds the n^{th} point of a spatial object. If you pass a negative number, zero, or a number larger than the total number of points on the linestring, `ST_PointN` returns NULL.

The vertex order is based on the Well-Known Text (WKT) representation of the spatial object.

Behavior Type

Immutable

Syntax

```
ST_PointN( g, n )
```

Arguments

<i>g</i>	Spatial object to search, type GEOMETRY or GEOGRAPHY
<i>n</i>	Point in the spatial object to be returned. The index is one-based, type INTEGER

Returns

GEOMETRY or GEOGRAPHY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how to use `ST_PointN`.

Returns the fifth point:

```
=> SELECT ST_AsText(ST_PointN(ST_GeomFromText('
      POLYGON(( 2 6, 2 9, 6 9, 7 7, 4 6, 2 6))'), 5));
   ST_AsText
-----
POINT (4 6)
(1 row)
```

Returns the second point:

```
=> SELECT ST_AsText(ST_PointN(ST_GeographyFromText('
      LINestring(23.41 24.93,34.2 32.98,40.7 41.19)'), 2));
   ST_AsText
-----
POINT (34.2 32.98)
(1 row)
```

ST_Relate

Determines if a given `GEOMETRY` object is spatially related to another `GEOMETRY` object, based on the specified DE-9IM pattern matrix string.

The DE-9IM standard identifies how two objects are spatially related to each other.

Behavior Type

Immutable

Syntax

```
ST_Relate( g1, g2, matrix )
```

Arguments

<i>g1</i>	Spatial object, type <code>GEOMETRY</code>
<i>g2</i>	Spatial object, type <code>GEOMETRY</code>

<i>matrix</i>	<p>DE-9IM pattern matrix string, type CHAR(9). This string represents a 3 x 3 matrix of restrictions on the dimensions of the respective intersections of the interior, boundary, and exterior of the two geometries. Must contain exactly 9 of the following characters:</p> <ul style="list-style-type: none">• T• F• 0• 1• 2• *
---------------	---

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

The following examples show how to use ST_Relate.

The DE-9IM pattern for "equals" is 'T*F**FFF2':

```
=> SELECT ST_Relate(ST_GeomFromText('LINESTRING(0 1,2 2)'),
  ST_GeomFromText('LINESTRING(2 2,0 1)'), 'T*F**FFF2');
  ST_Relate
-----
t
(1 row)
```

The DE-9IM pattern for "overlaps" is 'T*T***T**':

```
=> SELECT ST_Relate(ST_GeomFromText('POLYGON((-1 -1,0 1,2 2,-1 -1))'),
  ST_GeomFromText('POLYGON((0 1,1 -1,1 1,0 1))'), 'T*T***T**');
  ST_Relate
-----
t
(1 row)
```

ST_SRID

Identifies the spatial reference system identifier (SRID) stored with a spatial object.

The SRID of a GEOMETRY object can only be determined when passing an SRID to either ST_GeomFromText or ST_GeomFromWKB. ST_SRID returns this stored value. SRID values of 0 to $2^{32}-1$ are valid.

Behavior Type

Immutable

Syntax

```
ST_SRID( g )
```

Arguments

<i>g</i>	Spatial object for which you want the SRID, type GEOMETRY or GEOGRAPHY
----------	--

Returns

INTEGER

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	Yes	No	No

Examples

The following examples show how to use ST_SRID.

The default SRID of a GEOMETRY object is 0:

```
=> SELECT ST_SRID(ST_GeomFromText(
      'POLYGON((-1 -1,2 2,0 1,-1 -1))');
ST_SRID
-----
0
(1 row)
```

The default SRID of a GEOGRAPHY object is 4326:

```
=> SELECT ST_SRID(ST_GeographyFromText(
      'POLYGON((22 35,24 35,26 32,22 35))');
ST_SRID
-----
4326
(1 row)
```

ST_SymDifference

Calculates all the points in two GEOMETRY objects except for the points they have in common, but including the boundaries of both objects.

This result is called the symmetric difference and is represented mathematically as: $\text{Closure}(g1 - g2) \cup \text{Closure}(g2 - g1)$

Behavior Type

Immutable

Syntax

```
ST_SymDifference( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Returns

GEOMETRY

Supported Data Types

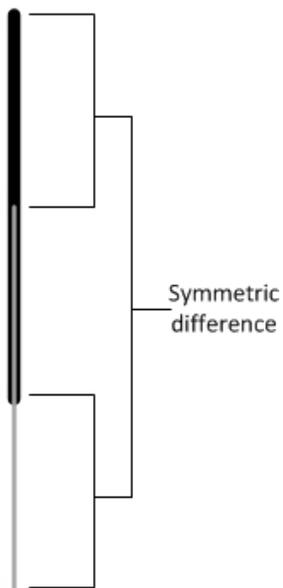
Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes

Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

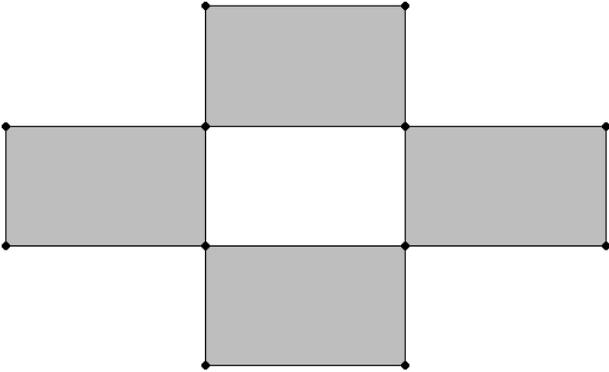
The following examples show how to use ST_SymDifference.

Returns the two linestrings:



```
=> SELECT ST_AsText(ST_SymDifference(ST_GeomFromText('LINESTRING(30 40,
30 55)'),ST_GeomFromText('LINESTRING(30 32.5,30 47.5)'));
ST_AsText
-----
MULTILINESTRING ((30 47.5, 30 55),(30 32.5,30 40))
(1 row)
```

Returns four squares:



```
=> SELECT ST_AsText(ST_SymDifference(ST_GeomFromText('POLYGON((2 1,2 4,3 4,
  3 1,2 1))'),ST_GeomFromText('POLYGON((1 2,1 3,4 3,4 2,1 2))'));
      ST_AsText
-----
MULTIPOLYGON (((2 1, 2 2, 3 2, 3 1, 2 1)), ((1 2, 1 3, 2 3, 2 2, 1 2)),
((2 3, 2 4, 3 4, 3 3, 2 3)), ((3 2, 3 3, 4 3, 4 2, 3 2)))
(1 row)
```

ST_Touches

Determines if two GEOMETRY objects touch at a single point or along a boundary, but do not have interiors that intersect.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable

Syntax

```
ST_Touches( g1, g2
            [USING PARAMETERS spheroid={true | false}] )
```

Arguments

<i>g1</i>	Spatial object, value of type GEOMETRY
<i>g2</i>	Spatial object, value of type GEOMETRY

Parameters

spheroid = {true false}	(Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False
---------------------------	--

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (WGS84)
Point	Yes	Yes
Multipoint	Yes	No
Linestring	Yes	No
Multilinestring	Yes	No
Polygon	Yes	Yes
Multipolygon	Yes	No
GeometryCollection	Yes	No

Compatible GEOGRAPHY pairs:

Data Type	GEOGRAPHY (WGS84)
Point-Point	No
Linestring-Point	No
Polygon-Point	Yes
Multipolygon-Point	No

Examples

The following examples show how to use `ST_Touches`.

Two polygons touch at a single point:

```
=> SELECT ST_Touches(ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
  ST_GeomFromText('POLYGON((1 3,0 3,1 2,1 3))'));
  ST_Touches
-----
t
(1 row)
```

Two polygons touch only along part of the boundary:

```
=> SELECT ST_Touches(ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
  ST_GeomFromText('POLYGON((1 2,0 3,0 1,1 2))'));
  ST_Touches
-----
t
(1 row)
```

Two polygons do not touch at any point:

```
=> SELECT ST_Touches(ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
  ST_GeomFromText('POLYGON((0 2,-1 3,-2 0,0 2))'));
  ST_Touches
-----
f
(1 row)
```

ST_Transform

Returns a new `GEOMETRY` with its coordinates converted to the spatial reference system identifier (SRID) used by the `srid` argument.

This function supports the following transformations:

- EPSG 4326 (WGS84) to EPSG 3857 (Web Mercator)
- EPSG 3857 (Web Mercator) to EPSG 4326 (WGS84)

For EPSG 4326 (WGS84), unless the coordinates fall within the following ranges, conversion results in failure:

- Longitude limits: -572 to +572
- Latitude limits: -89.9999999 to +89.9999999

Behavior Type

Immutable

Syntax

```
ST_Transform( g1, srid )
```

Arguments

<i>g1</i>	Spatial object of type GEOMETRY.
<i>srid</i>	Spatial reference system identifier (SRID) to which you want to convert your spatial object, of type INTEGER.

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	No	No
Multipoint	Yes	No	No
Linestring	Yes	No	No
Multilinestring	Yes	No	No
Polygon	Yes	No	No

Multipolygon	Yes	No	No
GeometryCollection	Yes	No	No

Examples

The following example shows how you can transform data from Web Mercator (3857) to WGS84 (4326):

```
=> SELECT ST_AsText(ST_Transform(STV_GeometryPoint(7910240.56433, 5215074.23966, 3857), 4326));
           ST_AsText
-----
POINT (71.0589 42.3601)
(1 row)
```

The following example shows how you can transform linestring data in a table from WGS84 (4326) to Web Mercator (3857):

```
=> CREATE TABLE transform_line_example (g GEOMETRY);
CREATE TABLE
=> COPY transform_line_example (gx FILLER LONG VARCHAR, g AS ST_GeomFromText(gx, 4326)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> LINESTRING(0 0, 1 1, 2 2, 3 4)
>> \.
=> SELECT ST_AsText(ST_Transform(g, 3857)) FROM transform_line_example;
           ST_AsText
-----
LINESTRING (0 -7.08115455161e-10, 111319.490793 111325.142866, 222638.981587 222684.208506,
333958.47238 445640.109656)
(1 row)
```

The following example shows how you can transform point data in a table from WGS84 (4326) to Web Mercator (3857):

```
=> CREATE TABLE transform_example (x FLOAT, y FLOAT, srid INT);
CREATE TABLE
=> COPY transform_example FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 42.3601|71.0589|4326
>> 122.4194|37.7749|4326
>> 94.5786|39.0997|4326
>> \.
=> SELECT ST_AsText(ST_Transform(STV_GeometryPoint(x, y, srid), 3857)) FROM transform_example;
           ST_AsText
-----
POINT (4715504.76195 11422441.5961)
POINT (13627665.2712 4547675.35434)
POINT (10528441.5919 4735962.8206)
(3 rows)
```

ST_Union

Calculates the union of all points in two spatial objects.

This result is represented mathematically by: $g1 \cup g2$

Behavior Type

Immutable

Syntax

```
ST_Union( g1, g2 )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY
<i>g2</i>	Spatial object, type GEOMETRY

Returns

GEOMETRY

Supported Data Types

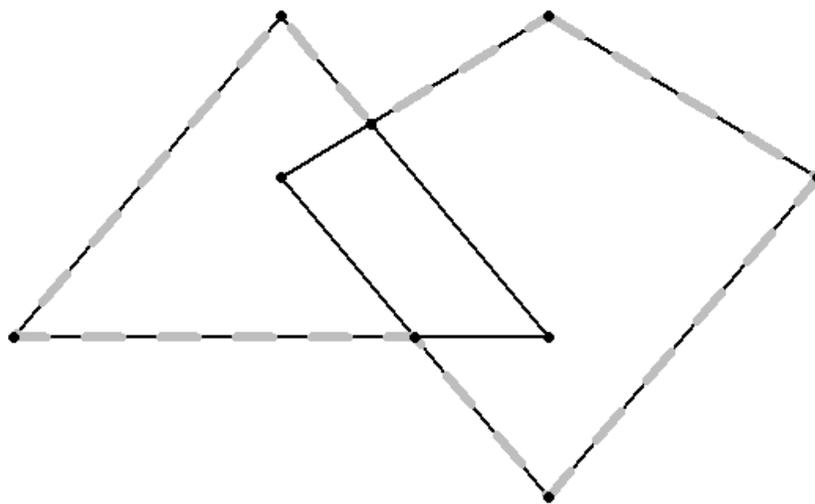
Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes

Multipolygon	Yes
GeometryCollection	Yes

Example

The following example shows how to use `ST_Union`.

Returns a polygon that represents all the points contained in these two polygons:



```
=> SELECT ST_AsText(ST_Union(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,-1 1,0 2))'),
  ST_GeomFromText('POLYGON((-1 2, 0 0, -2 0, -1 2)'))));
      ST_AsText
-----
POLYGON ((0 2, 1 1, 0 -1, -0.5 0, -2 0, -1 2, -0.6666666666667 1.333333333333, 0 2))
(1 row)
```

ST_Within

If spatial object `g1` is completely inside of spatial object `g2`, then `ST_Within` returns true. Both parameters must be the same spatial data type. Either specify two `GEOMETRY` objects or two `GEOGRAPHY` objects.

If an object such as a point or linestring only exists along a polygon's boundary, then `ST_Within` returns false. The interior of a linestring is all the points along the linestring except the start and end points.

`ST_Within(g1, g2)` is functionally equivalent to `ST_Contains(g2, g1)`.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable

Syntax

```
ST_Within( g1, g2  
          [USING PARAMETERS spheroid={true | false}] )
```

Arguments

<i>g1</i>	Spatial object, type GEOMETRY or GEOGRAPHY
<i>g2</i>	Spatial object, type GEOMETRY or GEOGRAPHY

Parameters

spheroid = {true false}	(Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False
---------------------------	--

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes

Multipoint	Yes	No	No
Linestring	Yes	Yes	No
Multilinestring	Yes	No	No
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	No
GeometryCollection	Yes	No	No

Compatible GEOGRAPHY pairs:

Data Type	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point-Point	Yes	No
Point-Linestring	Yes	No
Point-Polygon	Yes	Yes
Point-Multipolygon	Yes	No

Examples

The following examples show how to use ST_Within.

The first polygon is completely contained within the second polygon:

```
=> SELECT ST_Within(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
  ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
  ST_Within
-----
t
(1 row)
```

The point is on a vertex of the polygon, but not in its interior:

```
=> SELECT ST_Within (ST_GeographyFromText('POINT(30 25)'),
  ST_GeographyFromText('POLYGON((25 25,25 35,32.2 35,30 25,25 25))'));
  ST_Within
-----
f
(1 row)
```

Two polygons are spatially equivalent:

```
=> SELECT ST_Within (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),  
    ST_GeomFromText('POLYGON((0 3, -1 2, 0 1, 0 3))'));  
    ST_Within  
-----  
    t  
(1 row)
```

See Also

- [ST_Contains](#)
- [ST_Overlaps](#)

ST_X

Determines the *x*- coordinate for a GEOMETRY point or the longitude value for a GEOGRAPHY point.

Behavior Type

Immutable

Syntax

```
ST_X( g )
```

Arguments

<i>g</i>	Point of type GEOMETRY or GEOGRAPHY
----------	-------------------------------------

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	No	No	No
Linestring	No	No	No
Multilinestring	No	No	No
Polygon	No	No	No
Multipolygon	No	No	No
GeometryCollection	No	No	No

Examples

The following examples show how to use ST_X.

Returns the x-coordinate:

```
=> SELECT ST_X(ST_GeomFromText('POINT(3.4 1.25)'));
   ST_X
-----
    3.4
(1 row)
```

Returns the longitude value:

```
=> SELECT ST_X(ST_GeographyFromText('POINT(25.34 45.67)'));
   ST_X
-----
    25.34
(1 row)
```

ST_XMax

Returns the maximum x-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Place computes maximum coordinates by calculating the maximum longitude of the great circle arc from (MAX(longitude), ST_YMin(GEOGRAPHY)) to (MAX(longitude), ST_YMax(GEOGRAPHY)). In this case, MAX(longitude) is the maximum longitude value of the geography object.

If either latitude or longitude is out of range, ST_XMax returns the maximum plain value of the geography object.

Behavior Type

Immutable

Syntax

```
ST_XMax( g )
```

Arguments

<i>g</i>	Spatial object for which you want to find the maximum x-coordinate, type GEOMETRY or GEOGRAPHY.
----------	---

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes

Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Examples

The following examples show how to use ST_XMax.

Returns the maximum x-coordinate within a rectangle:

```
=> SELECT ST_XMax(ST_GeomFromText('POLYGON((0 1,0 2,1 2,1 1,0 1))'));
   ST_XMax
-----
         1
(1 row)
```

Returns the maximum longitude value within a rectangle:

```
=> SELECT ST_XMax(ST_GeographyFromText(
  'POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35))');
   ST_XMax
-----
        -71
(1 row)
```

ST_XMin

Returns the minimum x-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Place computes minimum coordinates by calculating the minimum longitude of the great circle arc from (MIN(longitude), ST_YMin(GEOGRAPHY)) to (MIN(longitude), ST_YMax(GEOGRAPHY)). In this case, MIN(longitude) represents the minimum longitude value of the geography object

If either latitude or longitude is out of range, ST_XMin returns the minimum plain value of the geography object.

Behavior Type

Immutable

Syntax

`ST_XMin(g)`

Arguments

<i>g</i>	Spatial object for which you want to find the minimum <i>x</i> -coordinate, type GEOMETRY or GEOGRAPHY.
----------	---

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Examples

The following examples show how to use ST_XMin.

Returns the minimum *x*-coordinate within a rectangle:

```
=> SELECT ST_XMin(ST_GeomFromText('POLYGON((0 1,0 2,1 2,1 1,0 1))'));
   ST_XMin
-----
        0
(1 row)
```

Returns the minimum longitude value within a rectangle:

```
=> SELECT ST_XMin(ST_GeographyFromText(
   'POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35))');
   ST_XMin
-----
     -71.5
(1 row)
```

ST_YMax

Returns the maximum *y*-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Place computes maximum coordinates by calculating the maximum latitude of the great circle arc from (ST_XMin(GEOGRAPHY), MAX(latitude)) to (ST_XMax(GEOGRAPHY), MAX(latitude)). In this case, MAX(latitude) is the maximum latitude value of the geography object.

If either latitude or longitude is out of range, ST_YMax returns the maximum plain value of the geography object.

Behavior Type

Immutable

Syntax

```
ST_YMax( g )
```

Arguments

<i>g</i>	Spatial object for which you want to find the maximum <i>y</i> -coordinate, type GEOMETRY or GEOGRAPHY.
----------	---

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Examples

The following examples show how to use ST_YMax.

Returns the maximum *y*-coordinate within a rectangle:

```
=> SELECT ST_YMax(ST_GeomFromText('POLYGON((0 1,0 4,1 4,1 1,0 1))'));
   ST_YMax
-----
         4
(1 row)
```

Returns the maximum latitude value within a rectangle:

```
=> SELECT ST_YMax(ST_GeographyFromText(
  'POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35))');
   ST_YMax
-----
42.3802715689979
(1 row)
```

ST_YMin

Returns the minimum *y*-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Place computes minimum coordinates by calculating the minimum latitude of the great circle arc from (ST_XMin(GEOGRAPHY), MIN(latitude)) to (ST_XMax(GEOGRAPHY), MIN(latitude)). In this case, MIN(latitude) represents the minimum latitude value of the geography object.

If either latitude or longitude is out of range, ST_YMin returns the minimum plain value of the geography object.

Behavior Type

Immutable

Syntax

```
ST_YMin( g )
```

Arguments

<i>g</i>	Spatial object for which you want to find the minimum <i>y</i> -coordinate, type GEOMETRY or GEOGRAPHY.
----------	---

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes

Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Examples

The following examples show how to use ST_YMin.

Returns the minimum y-coordinate within a rectangle:

```
=> SELECT ST_YMin(ST_GeomFromText('POLYGON((0 1,0 4,1 4,1 1,0 1))'));
   ST_YMin
-----
         1
(1 row)
```

Returns the minimum latitude value within a rectangle:

```
=> SELECT ST_YMin(ST_GeographyFromText(
  'POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35))');
   ST_YMin
-----
    42.35
(1 row)
```

ST_Y

Determines the y-coordinate for a GEOMETRY point or the latitude value for a GEOGRAPHY point.

Behavior Type

Immutable

Syntax

`ST_Y(g)`

Arguments

<i>g</i>	Point of type GEOMETRY or GEOGRAPHY
----------	-------------------------------------

Returns

FLOAT

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	No	No	No
Linestring	No	No	No
Multilinestring	No	No	No
Polygon	No	No	No
Multipolygon	No	No	No
GeometryCollection	No	No	No

Examples

The following examples show how to use `ST_Y`.

Returns the *y*-coordinate:

```
=> SELECT ST_Y(ST_GeomFromText('POINT(3 5.25)'));
   ST_Y
-----
   5.25
(1 row)
```

Returns the latitude value:

```
=> SELECT ST_Y(ST_GeographyFromText('POINT(35.44 51.04)'));
   ST_Y
-----
   51.04
(1 row)
```

STV_AsGeoJSON

Returns the geometry or geography argument as a Geometry Javascript Object Notation (GeoJSON) object.

Behavior Type

Immutable

Syntax

```
STV_AsGeoJSON(g, [USING PARAMETERS maxdecimals=dec_value])
```

Arguments

<i>g</i>	Spatial object of type GEOMETRY or GEOGRAPHY
maxdecimals = <i>dec_value</i>	(Optional) Integer value. Determines the maximum number of digits to output after the decimal of floating point coordinates. Valid values: Between 0 and 15. Default value: 6

Returns

LONG VARCHAR

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how you can use STV_AsGeoJSON.

Convert a geometry polygon to GeoJSON:

```
=> SELECT STV_AsGeoJSON(ST_GeomFromText('POLYGON((3 2, 4 3, 5 1, 3 2), (3.5 2, 4 2.5, 4.5 1.5, 3.5 2))'));
          STV_AsGeoJSON
-----
{"type": "Polygon", "coordinates": [[[3,2],[4,3],[5,1],[3,2]],[[3.5,2],[4,2.5],[4.5,1.5],[3.5,2]]]}
(1 row)
```

Convert a geography point to GeoJSON:

```
=> SELECT STV_AsGeoJSON(ST_GeographyFromText('POINT(42.36011 71.05899)') USING PARAMETERS
maxdecimals=4);
          STV_AsGeoJSON
-----
{"type": "Point", "coordinates": [42.3601,71.059]}
(1 row)
```

STV_Create_Index

Creates a spatial index on a set of polygons to speed up spatial intersection with a set of points.

A spatial index is created from an input polygon set, which can be the result of a query. Spatial indexes are created in a global name space. Vertica uses a distributed plan whenever the input table or projection is segmented across nodes of the cluster.

The OVER() clause must be empty.

Important: You cannot access spatial indexes on newly added nodes without rebalancing your cluster. For more information, see [REBALANCE_CLUSTER](#).

Behavior Type

Immutable

Note: Indexes are not connected to any specific table. Subsequent DML commands on the underlying table or tables of the input data source do not modify the index.

Syntax

```
STV_Create_Index( gid, g
                 USING PARAMETERS index='index_name'
                               [, overwrite={ true | false } ]
                               [, max_mem_mb=mem_value]
                               [, skip_nonindexable_polygons={true | false } ] )
OVER()
[ AS (polygons, srid, min_x, min_y, max_x, max_y, info) ]
```

Arguments

<i>gid</i>	Name of an integer column that uniquely identifies the polygon. The gid cannot be NULL.
<i>g</i>	Name of a geometry or geography (WGS84) column or expression that contains polygons and multipolygons. Only polygon and multipolygon can be indexed. Other shape types are excluded from the index.

Parameters

<code>index = 'index_name'</code>	Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not
-----------------------------------	---

	allowed in index names.
<code>overwrite = [true false]</code>	<p>(Optional) BOOLEAN value that specifies whether to overwrite the index, if an index exists. This parameter cannot be NULL.</p> <p>Default: False</p>
<code>max_mem_mb = maxmem_value</code>	<p>(Optional) A positive integer that assigns a limit to the amount of memory in megabytes that <code>STV_Create_Index</code> can allocate during index construction. On a multi-node database this is the memory limit per node. The default value is 256. Do not assign a value higher than the amount of memory in the GENERAL resource pool. For more information about this pool, see Using Queries to Monitor Resource Pool Size and Usage.</p> <p>Setting a value for <code>max_mem_mb</code> that is at or near the maximum memory available on the node can negatively affect your system's performance. For example, it could cause other queries to time out waiting for memory resources during index construction.</p>
<code>skip_nonindexable_polygons = [true false]</code>	<p>(Optional) BOOLEAN</p> <p>In rare cases, intricate polygons (for instance, with too high resolution or anomalous spikes) cannot be indexed. These polygons are considered non-indexable. When set to False, non-indexable polygons cause the index creation to fail. When set to True, index creation can succeed by excluding non-indexable polygons from the index.</p> <p>To review the polygons that were not able to be indexed, use <code>STV_Describe_Index</code> with the parameter <code>list_polygon</code>.</p> <p>Default: False</p>

Returns

<i>polygons</i>	Number of polygons indexed.
<i>SRID</i>	Spatial reference system identifier.
<i>min_x, min_y, max_x, max_y</i>	Coordinates of the minimum bounding rectangle (MBR) of the indexed geometries. (<i>min_x, min_y</i>) are the south-west coordinates, and (<i>max_x, max_y</i>) are the north-east coordinates.
<i>info</i>	Lists the number of excluded spatial objects as well as their type that were excluded from the index.

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (WGS84)
Point	No	No
Multipoint	No	No
Linestring	No	No
Multilinestring	No	No
Polygon	Yes	Yes
Multipolygon	Yes	No
GeometryCollection	No	No

Privileges

Any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

Recommendations

- Segment large polygon tables across multiple nodes. Table segmentation causes index creation to run in parallel, leveraging the Massively Parallel Processing (MPP) architecture in Vertica. This significantly reduces execution time on large tables.

Vertica recommends that you segment the table from which you are building the index when the total number of polygons is large.

- `STV_Create_Index` can consume large amounts of processing time and memory.

Vertica recommends that when indexing new data for the first time, you monitor memory usage to be sure it stays within safe limits. Memory usage depends on number of polygons, number of vertices, and the amount of overlap among polygons.

- `STV_Create_Index` tries to allocate memory before it starts creating the index. If it cannot allocate enough memory, the function fails. If not enough memory is available, try the following:
 - Create the index at a time of less load on the system.
 - Avoid concurrent index creation.
 - Try segmenting the input table across the nodes of the cluster.
- Ensure that all of the polygons you plan to index are valid polygons. `STV_Create_Index` and `STV_Refresh_Index` do not check polygon validity when building an index.

For more information, see [Ensuring Polygon Validity Before Creating or Refreshing an Index](#).

Limitations

- Any indexes created prior to 8.1.x need to be re-created.
- Index creation fails if there are WGS84 polygons with vertices on the International Date Line (IDL) or the North and South Poles.
- The backslash or tab characters are not allowed in index names.
- Indexes cannot have names greater than 110 characters.

- The following geometries are excluded from the index:
 - Non-polygons
 - Geometries with NULL identifiers
 - NULL (multi) polygon
 - EMPTY (multi) polygon
 - Invalid (multi) polygon
- The following geographies are excluded from the index:
 - Polygons with holes
 - Polygons crossing the International Date Line
 - Polygons covering the north or south pole
 - Antipodal polygons

Usage Tips

- To cancel an STV_Create_Index run, use **Ctrl + C**.
- If there are no valid polygons in the geom column, STV_Create_Index reports an error in vertica.log and stops index creation.
- If index creation uses a large amount of memory, consider segmenting your data to utilize parallel index creation.

Examples

The following examples show how to use STV_Create_Index.

Create an index with a single literal argument:

```
=> SELECT STV_Create_Index(1, ST_GeomFromText('POLYGON((0 0,0 15.2,3.9 15.2,3.9 0,0 0))')
      USING PARAMETERS index='my_polygon') OVER();
polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----
        1 |    0 |    0 |    0 |    3.9 |   15.2 |
(1 row)
```

Create an index from a table:

```
=> CREATE TABLE polys (gid INT, geom GEOMETRY(1000));
CREATE TABLE
=> COPY polys(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POLYGON((-31 74,8 70,8 50,-36 53,-31 74))
>> 2|POLYGON((-38 50,4 13,11 45,0 65,-38 50))
>> 3|POLYGON((10 20,15 60,20 45,46 15,10 20))
>> 4|POLYGON((5 20,9 30,20 45,36 35,5 20))
>> 5|POLYGON((12 23,9 30,20 45,36 35,37 67,45 80,50 20,12 23))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true,
    max_mem_mb=256) OVER() FROM polys;
polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----
      5 |    0 |   -38 |    13 |    50 |    80 |
(1 row)
```

Create an index in parallel from a partitioned table:

```
=> CREATE TABLE polys (p INT, gid INT, geom GEOMETRY(1000)) SEGMENTED BY HASH(p) ALL NODES;
CREATE TABLE
=> COPY polys (p, gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|10|POLYGON((-31 74,8 70,8 50,-36 53,-31 74))
>> 1|11|POLYGON((-38 50,4 13,11 45,0 65,-38 50))
>> 3|12|POLYGON((-12 42,-12 42,27 48,14 26,-12 42))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons', overwrite=true,
    max_mem_mb=256) OVER() FROM polys;
polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----
      3 |    0 |   -38 |    13 |    27 |    74 |
(1 row)
```

See Also

- [Spatial Joins with ST_Intersects and STV_Intersect](#)
- [STV_Intersect Scalar Function](#)
- [STV_Intersect Transform Function](#)
- [STV_Describe_Index](#)
- [STV_Drop_Index](#)
- [STV_Rename_Index](#)
- [Ensuring Polygon Validity Before Creating or Refreshing an Index](#)

STV_Describe_Index

Retrieves information about an index that contains a set of polygons. If you do not pass any parameters, `STV_Describe_Index` returns all of the defined indexes.

The `OVER()` clause must be empty.

Behavior Type

Immutable

Syntax

```
STV_Describe_Index ( [ USING PARAMETERS [index='index_name']  
                    [, list_polygons={true | false } ] ] ) OVER ( )
```

Arguments

<code>index = 'index_name'</code>	Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not allowed in index names.
<code>list_polygon</code>	(Optional) BOOLEAN that specifies whether to list the polygons in the index. The index argument must be used with this argument.

Returns

<code>polygons</code>	Number of polygons indexed.
<code>SRID</code>	Spatial reference system identifier.
<code>min_x, min_y, max_x, max_y</code>	Coordinates of the minimum bounding rectangle (MBR) of the indexed geometries. (<code>min_x, min_y</code>) are the south-west coordinates, and (<code>max_x, max_y</code>) are the north-east coordinates.
<code>name</code>	The name of the spatial index(es).

<i>gid</i>	Name of an integer column that uniquely identifies the polygon. The gid cannot be NULL.
<i>state</i>	The spatial object's state in the index. Possible values are: <ul style="list-style-type: none"> INDEXED - The spatial object was successfully indexed. SELF_INTERSECT - (WGS84 Only) The spatial object was not indexed because one of its edges intersects with another of its edges. EDGE_CROSS_IDL - (WGS84 Only) The spatial object was not indexed because one of its edges crosses the International Date Line. EDGE_HALF_CIRCLE - (WGS84 Only) The spatial object was not indexed because it contains two adjacent vertices that are antipodal. NON_INDEXABLE - The spatial object was not able to be indexed.
<i>geography</i>	The Well-Known Binary (WKB) representation of the spatial object.
<i>geometry</i>	The Well-Known Binary (WKB) representation of the spatial object.

Privileges

Any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

Limitations

Some functionality will require the index to be rebuilt if the index was created with 7.0.x or earlier.

Examples

The following examples show how to use STV_Describe_Index.

Retrieve information about the index:

```
=> SELECT STV_Describe_Index (USING PARAMETERS index='my_polygons') OVER ();
   type | polygons | SRID | min_x | min_y | max_x | max_y
-----+-----+-----+-----+-----+-----+-----
GEOMETRY |      4 |    0 |   -1 |   -1 |    12 |    12
(1 row)
```

Return the names of all the defined indexes:

```
=> SELECT STV_Describe_Index() OVER ();
      name
-----
MA_counties_index
my_polygons
NY_counties_index
US_States_Index
(4 rows)
```

Return the polygons included in an index:

```
=> SELECT STV_Describe_Index(USING PARAMETERS index='my_polygons', list_polygons=TRUE) OVER ();
gid | state | geometry
-----+-----+-----
 12 | INDEXED | \260\000\000\000\000\000\000\ ...
 14 | INDEXED | \200\000\000\000\000\000\000\ ...
 10 | NON_INDEXABLE | \274\000\000\000\000\000\000\ ...
 11 | INDEXED | \260\000\000\000\000\000\000\ ...
(4 rows)
```

See Also

- [Spatial Joins with ST_Intersects and STV_Intersect](#)
- [STV_Intersect Scalar Function](#)
- [STV_Intersect Transform Function](#)
- [STV_Drop_Index](#)
- [STV_Rename_Index](#)

STV_Drop_Index

Deletes a spatial index. If `STV_Drop_Index` cannot find the specified spatial index, it returns an error.

The `OVER` clause must be empty.

Behavior Type

Immutable

Syntax

```
STV_Drop_Index( USING PARAMETERS index = 'index_name' ) OVER ( )
```

Arguments

<code>index = 'index_name'</code>	Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not allowed in index names.
-----------------------------------	---

Example

The following example shows how to use STV_Drop_Index.

Drop an index:

```
=> SELECT STV_Drop_Index(USING PARAMETERS index ='my_polygons') OVER ();
 drop_index
-----
Index dropped
(1 row)
```

See Also

- [Spatial Joins with ST_Intersects and STV_Intersect](#)
- [STV_Create_Index](#)
- [STV_Describe_Index](#)
- [STV_Rename_Index](#)
- [STV_Intersect Scalar Function](#)
- [STV_Intersect Transform Function](#)

STV_DWithin

Determines if the shortest distance from the boundary of one spatial object to the boundary of another object is within a specified distance.

Parameters *g1* and *g2* must be both GEOMETRY objects or both GEOGRAPHY objects.

Behavior Type

Immutable

Syntax

```
STV_DWithin( g1, g2, d )
```

Arguments

<i>g1</i>	Spatial object of type GEOMETRY or GEOGRAPHY
<i>g2</i>	Spatial object of type GEOMETRY or GEOGRAPHY
<i>d</i>	Value of type FLOAT indicating a distance. For GEOMETRY objects, the distance is measured in Cartesian coordinate units. For GEOGRAPHY objects, the distance is measured in meters.

Returns

BOOLEAN

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes

Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Compatible GEOGRAPHY pairs:

Data Type	GEOGRAPHY (Perfect Sphere)
Point-Point	Yes
Point-Linestring	Yes
Point-Polygon	Yes
Point-Multilinestring	Yes
Point-Multipolygon	Yes

Examples

The following examples show how to use STV_DWithin.

Two geometries are one Cartesian coordinate unit from each other at their closest points:

```
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((-1 -1,2 2,0 1,-1 -1)'),
  ST_GeomFromText('POLYGON((4 3,2 3,4 5,4 3)'),1);
  STV_DWithin
-----
t
(1 row)
```

If you reduce the distance to 0.99 units:

```
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((-1 -1,2 2,0 1,-1 -1)'),
  ST_GeomFromText('POLYGON((4 3,2 3,4 5,4 3)'),0.99);
  STV_DWithin
-----
f
(1 row)
```

The first polygon touches the second polygon:

```
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((-1 -1,2 2,0 1,-1 -1)'),
    ST_GeomFromText('POLYGON((1 1,2 3,4 5,1 1)'),0.00001);
    STV_DWithin
-----
t
(1 row)
```

The first polygon is not within 1000 meters from the second polygon:

```
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((45.2 40,50.65 51.29,
    55.67 47.6,50 47.6,45.2 40)'),ST_GeomFromText('POLYGON((25 25,25 30,
    30 30,30 25,25 25)'), 1000);
    STV_DWithin
-----
t
(1 row)
```

STV_Export2Shapefile

Exports GEOGRAPHY or GEOMETRY data from a database table or a subquery to a shapefile. Writes the output to the directory specified using [STV_SetExportShapefileDirectory](#).

Behavior Type

Immutable

Syntax

```
STV_Export2Shapefile( columns
                    USING PARAMETERS shapefile = 'name_of_shapefile'
                               [, overwrite = { TRUE | FALSE } ]
                               [, shape = ' { Point | Polygon | Linestring | Multipoint |
Multipolygon | Multilinestring } ' ] )
                    OVER()
```

Parameters

```
shapefile = 'name_of_shapefile'
```

Prefix of the component names of the shapefile, type VARCHAR. Must

	<p>end with the file extension <code>.shp</code>. Limited to 128 octets in length. For example, <code>city-data.shp</code>.</p> <p>If you want to save the shapefile to a sub-directory you can do so by concatenating the sub-directory to <code>name_of_shapefile</code>. For example, <code>visualizations/city-data.shp</code>.</p>
<pre>overwrite = { TRUE FALSE }</pre>	<p>(Optional) BOOLEAN value that specifies whether to overwrite the index, if an index exists. This parameter cannot be NULL.</p> <p>Default: False</p> <p>Overwriting may corrupt the existing files.</p>
<pre>shape = ' { Point Polygon Linestring Multipoint Multipolygon Multilinestring } '</pre>	<p>Must be one of the following spatial classes: Point, Polygon, Linestring, Multipoint, Multipolygon,</p>

	Multilinestring. Polygons and multipolygons always have a clockwise orientation. Default: Polygon
--	---

Arguments

<i>columns</i>	The columns to export to the shapefile. A value of asterisk (*) is the equivalent to listing all columns of the FROM clause.
----------------	---

Returns

Three files in the shapefile export directory with the extensions .shp, .shx, and .dbf.

Limitations

- If a multipolygon, multilinestring, or multipoint contains only one element, then it is written as a polygon, line, or point, respectively.
- Column names longer than 10 characters are truncated.
- Empty POINTS cannot be exported.
- All rows with NULL geometry or geography data are skipped.
- Unsupported or invalid dates are replaced with NULLs.
- Numeric values may lose precision when they are exported. This loss occurs because the target field in the .dbf file is a 64-bit FLOAT column, which can only represent about 15 significant digits.

Examples

The following example shows how you can use `STV_Export2Shapefile` to export all columns from the table `geo_data` to a shapefile named `city-data.shp`:

```
=> SELECT STV_Export2Shapefile(*
        USING PARAMETERS shapefile = 'visualizations/city-data.shp',
                          overwrite = true, shape = 'Point')
        OVER()
        FROM geo_data
        WHERE REVENUE > 25000;
Rows Exported | File Path
-----+-----
6442892 | v_geo-db_node0001: /home/geo/temp/visualizations/city-data.shp
(1 row)
```

STV_Extent

Returns a bounding box containing all of the input data.

Use `STV_Extent` inside of a nested query for best results. The `OVER` clause must be empty.

Important: `STV_Extent` does not return a valid polygon when the input is a single point.

Behavior Type

Immutable

Syntax

```
STV_Extent( g )
```

Arguments

<i>g</i>	Spatial object, type GEOMETRY.
----------	--------------------------------

Returns

GEOMETRY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	Yes

Examples

The following examples show how you can use STV_Extent.

Return the bounding box of a linestring, and verify that it is a valid polygon:

```
=> SELECT ST_AsText(geom) AS bounding_box, ST_IsValid(geom)
      FROM (SELECT STV_Extent(ST_GeomFromText('LineString(0 0, 1 1)')) OVER() AS geom) AS g;
      bounding_box          | ST_IsValid
-----+-----
POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)) | t
(1 row)
```

Return the bounding box of spatial objects in a table:

```
=> CREATE TABLE misc_geo_shapes (id IDENTITY, geom GEOMETRY);
CREATE TABLE
=> COPY misc_geo_shapes (gx FILLER LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> POINT(-71.03 42.37)
>> LINESTRING(-71.058849 42.367501, -71.062240 42.371276, -71.067938 42.371246)
>> POLYGON((-71.066030 42.380617, -71.055827 42.376734, -71.060811 42.376011, -71.066030 42.380617))
>> \.
=> SELECT ST_AsText(geom_col) AS bounding_box
      FROM (SELECT STV_Extent(geom) OVER() AS geom_col FROM misc_geo_shapes) AS g;
      bounding_box
-----
-----
```

```
POLYGON ((-71.067938 42.367501, -71.03 42.367501, -71.03 42.380617, -71.067938 42.380617, -71.067938 42.367501))  
(1 row)
```

STV_ForceLHR

Alters the order of the vertices of a spatial object to follow the left-hand-rule.

Behavior Type

Immutable

Syntax

```
STV_ForceLHR( g, [USING PARAMETERS skip_nonreorientable_polygons={true | false} ])
```

Arguments

<i>g</i>	Spatial object, type GEOGRAPHY.
<code>skip_nonreorientable_polygons = { true false }</code>	<p>(Optional) Boolean</p> <p>When set to False, non-orientable polygons generate an error. For example, if you use <code>STV_ForceLHR</code> or <code>STV_Reverse</code> with <code>skip_nonorientable_polygons</code> set to False, a geography polygon containing a hole generates an error. When set to True, the result returned is the polygon, as passed to the API, without alteration.</p> <p>This argument can help you when you are creating an index from a table containing polygons that cannot be re-oriented.</p> <p>Vertica Place considers these polygons non-orientable:</p> <ul style="list-style-type: none">• Polygons with a hole

	<ul style="list-style-type: none"> • Multipolygons • Multipolygons with a hole <p>Default value: False</p>
--	---

Returns

GEOGRAPHY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	No	No	No
Multipoint	No	No	No
Linestring	No	No	No
Multilinestring	No	No	No
Polygon	No	Yes	Yes
Multipolygon	No	Yes	Yes
GeometryCollection	No	No	No

Examples

The following example shows how you can use STV_ForceLHR.

Re-orient a geography polygon to left-hand orientation:

```
=> SELECT ST_AsText(STV_ForceLHR(ST_GeographyFromText('Polygon((1 1, 3 1, 2 2, 1 1))')));
      ST_AsText
-----
POLYGON ((1 1, 3 1, 2 2, 1 1))
(1 row)
```

Reverse the orientation of a geography polygon by forcing left-hand orientation:

```
=> SELECT ST_AsText(STV_ForceLHR(ST_GeographyFromText('Polygon((1 1, 2 2, 3 1, 1 1))')));
       ST_AsText
-----
POLYGON ((1 1, 3 1, 2 2, 1 1))
(1 row)
```

See Also

[STV_Reverse](#)

STV_Geography

Casts a GEOMETRY object into a GEOGRAPHY object. The SRID value does not affect the results of Vertica Place queries.

When STV_Geography converts a GEOMETRY object to a GEOGRAPHY object, it sets its SRID to 4326.

Behavior Type

Immutable

Syntax

```
STV_Geography( geom )
```

Arguments

<i>geom</i>	Spatial object that you want to cast into a GEOGRAPHY object, type GEOMETRY
-------------	---

Returns

GEOGRAPHY

Supported Data Types

Data Type	GEOMETRY
Point	Yes
Multipoint	Yes
Linestring	Yes
Multilinestring	Yes
Polygon	Yes
Multipolygon	Yes
GeometryCollection	No

Example

The following example shows how to use STV_Geography.

To calculate the centroid of the GEOGRAPHY object, convert it to a GEOMETRY object, then convert it back to a GEOGRAPHY object:

```
=> CREATE TABLE geogs(g GEOGRAPHY);
CREATE TABLE
=> COPY geogs(gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> MULTIPOINT(-108.619726 45.000284,-107.866813 45.00107,-106.363711 44.994223,-70.847746 41.205814)
>> \.
=> SELECT ST_AsText(STV_Geography(ST_Centroid(STV_Geometry(g)))) FROM geogs;
      ST_AsText
-----
POINT (-98.424499 44.05034775)
(1 row)
```

STV_GeographyPoint

Returns a GEOGRAPHY point based on the input values.

This is the optimal way to convert raw coordinates to GEOGRAPHY points.

Behavior Type

Immutable

Syntax

```
STV_GeographyPoint( x, y )
```

Arguments

x	x-coordinate or longitude, FLOAT.
y	y-coordinate or latitude, FLOAT.

Returns

GEOGRAPHY

Examples

The following examples show how to use STV_GeographyPoint.

Return a GEOGRAPHY point:

```
=> SELECT ST_AsText(STV_GeographyPoint(-114.101588, 47.909677));
           ST_AsText
-----
POINT (-114.101588 47.909677)
(1 row)
```

Return GEOGRAPHY points using two columns:

```
=> CREATE TABLE geog_data (id IDENTITY, x FLOAT, y FLOAT);
CREATE TABLE
=> COPY geog_data FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> -114.101588|47.909677
>> -111.532377|46.430753
>> \.
=> SELECT id, ST_AsText(STV_GeographyPoint(x, y)) FROM geog_data;
id |           ST_AsText
```

```
-----+-----  
 1 | POINT (-114.101588 47.909677)  
 2 | POINT (-111.532377 46.430753)  
(2 rows)
```

Create GEOGRAPHY points by manipulating data source columns during load:

```
=> CREATE TABLE geog_data_load (id IDENTITY, geog GEOGRAPHY);  
CREATE TABLE  
=> COPY geog_data_load (lon FILLER FLOAT,  
                        lat FILLER FLOAT,  
                        geog AS STV_GeographyPoint(lon, lat))  
FROM 'test_coords.csv' DELIMITER ',';  
Rows Loaded  
-----  
      2  
(1 row)  
=> SELECT id, ST_AsText(geog) FROM geog_data_load;  
id |          ST_AsText  
-----+-----  
 1 | POINT (-75.101654451 43.363830536)  
 2 | POINT (-75.106444487 43.367093798)  
(2 rows)
```

See Also

[STV_GeometryPoint](#)

STV_Geometry

Casts a GEOGRAPHY object into a GEOMETRY object.

The SRID value does not affect the results of Vertica Place queries.

Behavior Type

Immutable

Syntax

```
STV_Geometry( geog )
```

Arguments

<i>geog</i>	Spatial object that you want to cast into a GEOMETRY object, type GEOGRAPHY
-------------	---

Returns

GEOMETRY

Supported Data Types

Data Type	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	No	No

Example

The following example shows how to use STV_Geometry.

Convert the GEOGRAPHY values to GEOMETRY values, then convert the result back to a GEOGRAPHY type:

```
=> CREATE TABLE geogs(g GEOGRAPHY);
CREATE TABLE
=> COPY geogs(gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> MULTIPOINT(-108.619726 45.000284,-107.866813 45.00107,-106.363711 44.994223,-70.847746 41.205814)
>> \.
=> SELECT ST_AsText(STV_Geography(ST_Centroid(STV_Geometry(g)))) FROM geogs;
      ST_AsText
```

```
-----  
POINT (-98.424499 44.05034775)
```

STV_GeometryPoint

Returns a GEOMETRY point, based on the input values.

This approach is the most-optimal way to convert raw coordinates to GEOMETRY points.

Behavior Type

Immutable

Syntax

```
STV_GeometryPoint( x, y [, srid] )
```

Arguments

<i>x</i>	x-coordinate or longitude, FLOAT.
<i>y</i>	y-coordinate or latitude, FLOAT.
<i>srid</i>	(Optional) Spatial Reference Identifier (SRID) assigned to the point, INT.

Returns

GEOMETRY

Examples

The following examples show how to use STV_GeometryPoint.

Return a GEOMETRY point with an SRID:

```
=> SELECT ST_AsText(STV_GeometryPoint(71.148562, 42.989374, 4326));  
      ST_AsText
```

```
POINT (-71.148562 42.989374)
(1 row)
```

Return GEOMETRY points using two columns:

```
=> CREATE TABLE geom_data (id IDENTITY, x FLOAT, y FLOAT, SRID int);
CREATE TABLE
=> COPY geom_data FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 42.36383053600048|-71.10165445099966|4326
>> 42.3670937980005|-71.10644448699964|4326
>> \.
=> SELECT id, ST_AsText(STV_GeometryPoint(x, y, SRID)) FROM geom_data;
 id |          ST_AsText
-----+-----
  1 | POINT (-71.101654451 42.363830536)
  2 | POINT (-71.106444487 42.367093798)
(2 rows)
```

Create GEOMETRY points by manipulating data source columns during load:

```
=> CREATE TABLE geom_data_load (id IDENTITY, geom GEOMETRY);
CREATE TABLE
=> COPY geom_data_load (lon FILLER FLOAT,
                       lat FILLER FLOAT,
                       geom AS STV_GeometryPoint(lon, lat))
FROM 'test_coords.csv' DELIMITER ',';
Rows Loaded
-----
      2
(1 row)
=> SELECT id, ST_AsText(geom) FROM geom_data_load;
 id |          ST_AsText
-----+-----
  1 | POINT (-75.101654451 43.363830536)
  2 | POINT (-75.106444487 43.367093798)
(2 rows)
```

See Also

[STV_GeographyPoint](#)

STV_GetExportShapefileDirectory

Returns the path of the export directory.

Behavior Type

Immutable

Syntax

```
STV_GetExportShapefileDirectory( )
```

Returns

The path of the shapefile export directory.

Examples

The following example shows how you can use `STV_GetExportShapefileDirectory` to query the path of the shapefile export directory:

```
=> SELECT STV_GetExportShapefileDirectory();
       STV_GetExportShapefileDirectory
-----
Shapefile export directory: [/home/user/temp]
(1 row)
```

STV_Intersect Scalar Function

Spatially intersects a point or points with a set of polygons. The `STV_Intersect` scalar function returns the identifier associated with an intersecting polygon.

Behavior Type

Immutable

Syntax

```
STV_Intersect( { g | x , y }
              USING PARAMETERS index= 'index_name')
```

Arguments

<i>g</i>	A geometry or geography (WGS84) column that contains points. The <i>g</i> column can contain only point geometries or geographies. If the column contains a different geometry or geography type, STV_Intersect terminates with an error.
<i>x</i>	x-coordinate or longitude, FLOAT.
<i>y</i>	y-coordinate or latitude, FLOAT.

Parameters

<code>index = 'index_name'</code>	Name of the spatial index, of type VARCHAR.
-----------------------------------	---

Returns

The identifier of a matching polygon. If the point does not intersect any of the index's polygons, then the STV_Intersect scalar function returns NULL.

Examples

The following examples show how you can use STV_Intersect scalar.

Using two floats, return the gid of a matching polygon or NULL:

```
=> CREATE TABLE polys (gid INT, geom GEOMETRY(1000));
CREATE TABLE
=> COPY polys(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POLYGON((31 74,8 70,8 50,36 53,31 74))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true,
                           max_mem_mb=256) OVER() FROM polys;
   type | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |      1 |    0 |    8 |   50 |   36 |   74 |
(1 row)

=> SELECT STV_Intersect(12.5683, 55.6761 USING PARAMETERS index = 'my_polygons_1');
STV_Intersect
-----
```

```
1
(1 row)
```

Using a GEOMETRY column, return the gid of a matching polygon or NULL:

```
=> CREATE TABLE polygons (gid INT, geom GEOMETRY(700));
CREATE TABLE
=> COPY polygons (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POLYGON((-31 74,8 70,8 50,-36 53,-31 74))
>> 2|POLYGON((-38 50,4 13,11 45,0 65,-38 50))
>> 3|POLYGON((-18 42,-10 65,27 48,14 26,-18 42))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons', overwrite=true,
      max_mem_mb=256) OVER() FROM polygons;
  type | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |      3 |    0 |   -38 |    13 |    27 |    74 |
(1 row)

=> CREATE TABLE points (gid INT, geom GEOMETRY(700));
CREATE TABLE
=> COPY points (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 100|POINT(-1 52)
>> 101|POINT(-20 0)
>> 102|POINT(-8 25)
>> 103|POINT(0 0)
>> 104|POINT(1 5)
>> 105|POINT(20 45)
>> 106|POINT(-20 5)
>> 107|POINT(-20 1)
>> \.
=> SELECT gid AS pt_gid, STV_Intersect(geom USING PARAMETERS index='my_polygons') AS pol_gid
      FROM points ORDER BY pt_gid;
 pt_gid | pol_gid
-----+-----
    100 |      1
    101 |
    102 |      2
    103 |
    104 |
    105 |      3
    106 |
    107 |
(8 rows)
```

See Also

- [Best Practices for Spatial Joins](#)
- [STV_Intersect: Scalar Function vs. Transform Function](#)

- [STV_Intersect Transform Function](#)
- [STV_Create_Index](#)

STV_Intersect Transform Function

Spatially intersects points and polygons. The STV_Intersect transform function returns a tuple with matching point/polygon pairs. For every point, Vertica returns either one or many matching polygons.

You can improve performance when you parallelize the computation of the STV_Intersect transform function over multiple nodes. To parallelize the computation, use an OVER (PARTITION BEST) clause.

Behavior Type

Immutable

Syntax

```
STV_Intersect ( { gid | i }, { g | x , y }  
              USING PARAMETERS index='index_name')  
              OVER() AS (pt_gid, pol_gid)
```

Arguments

<i>gid i</i>	An integer column or integer that uniquely identifies the spatial object(s) of <i>g</i> or <i>x</i> and <i>y</i> .
<i>g</i>	A geometry or geography (WGS84) column that contains points. The <i>g</i> column can contain only point geometries or geographies. If the column contains a different geometry or geography type, STV_Intersect terminates with an error.
<i>x</i>	x-coordinate or longitude, FLOAT.
<i>y</i>	y-coordinate or latitude, FLOAT.

Parameters

<code>index = 'index_name'</code>	Name of the spatial index, of type VARCHAR.
-----------------------------------	---

Returns

<code>pt_gid</code>	Unique identifier of the point geometry or geography, of type INTEGER.
<code>pol_gid</code>	Unique identifier of the polygon geometry or geography, of type INTEGER.

Examples

The following examples show how you can use STV_Intersect transform.

Using two floats, return the matching point-polygon pairs.

```
=> CREATE TABLE pols (gid INT, geom GEOMETRY(1000));
CREATE TABLE
=> COPY pols(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POLYGON((31 74,8 70,8 50,36 53,31 74))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true,
                           max_mem_mb=256) OVER() FROM pols;
   type | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |      1 |    0 |    8 |    50 |   36 |   74 |
(1 row)

=> SELECT STV_Intersect(56, 12.5683, 55.6761 USING PARAMETERS index = 'my_polygons_1') OVER();
 pt_gid | pol_gid
-----+-----
    56 |      1
(1 row)
```

Using a GEOMETRY column, return the matching point-polygon pairs.

```
=> CREATE TABLE polygons (gid int, geom GEOMETRY(700));
CREATE TABLE
=> COPY polygons (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin;
Enter data to be copied followed by a new line.
End with a backslash and a period on a line by itself.
>> 10|POLYGON((5 5, 5 10, 10 10, 10 5, 5 5))
>> 11|POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))
>> 12|POLYGON((1 1, 1 3, 3 3, 3 1, 1 1))
```

```

>> 14|POLYGON((-1 -1, -1 12, 12 12, 12 -1, -1 -1))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons', overwrite=true, max_mem_
mb=256)
      OVER() FROM polygons;
      type | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |         4 |     0 |    -1 |    -1 |     12 |     12 |
(1 row)

=> CREATE TABLE points (gid INT, geom GEOMETRY(700));
CREATE TABLE
=> COPY points (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POINT(9 9)
>> 2|POINT(0 1)
>> 3|POINT(2.5 2.5)
>> 4|POINT(0 0)
>> 5|POINT(1 5)
>> 6|POINT(1.5 1.5)
>> \.
=> SELECT STV_Intersect(gid, geom USING PARAMETERS index='my_polygons') OVER (PARTITION BEST)
      AS (point_id, polygon_gid)
      FROM points;
point_id | polygon_gid
-----+-----
         5 |          14
         1 |          14
         1 |          10
         4 |          14
         4 |          11
         6 |          12
         6 |          14
         6 |          11
         2 |          14
         2 |          11
         3 |          12
         3 |          14
(12 rows)

```

You can improve query performance by using the `STV_Intersect` transform function in a `WHERE` clause. Performance improves because this syntax eliminates all points that do not intersect polygons in the index.

Return the count of points that intersect with the polygon, where `gid = 14`:

```

=> SELECT COUNT(pt_id) FROM
      (SELECT STV_Intersect(gid, geom USING PARAMETERS index='my_polygons')
      OVER (PARTITION BEST) AS (pt_id, pol_id) FROM points)
      AS T WHERE pol_id = 14;
COUNT
-----
         6
(1 row)

```

See Also

- [Best Practices for Spatial Joins](#)
- [STV_Intersect: Scalar Function vs. Transform Function](#)
- [STV_Create_Index](#)
- [STV_Intersect Scalar Function](#)

STV_IsValidReason

Determines if a spatial object is well formed or valid. If the object is not valid, `STV_IsValidReason` returns a string that explains where the invalidity occurs.

A polygon or multipolygon is valid if all of the following are true:

- The polygon is closed; its start point is the same as its end point.
- Its boundary is a set of linestrings.
- The boundary does not touch or cross itself.
- Any polygons in the interior that do not have more than one point touching the boundary of the exterior polygon.

If you pass an invalid object to a Vertica Place function, the function fails or returns incorrect results. To determine if a polygon is valid, first run `ST_IsValid`. `ST_IsValid` returns `TRUE` if the polygon is valid, `FALSE` otherwise.

Important: `STV_IsValidReason` supports only polygon and multipolygon `GEOMETRY` data types.

Behavior Type

Immutable

Syntax

```
STV_IsValidReason( g )
```

Arguments

<i>g</i>	Geospatial object to test for validity, value of type GEOMETRY or GEOGRAPHY (WGS84).
----------	--

Returns

LONG VARCHAR

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	Yes	No	No
Multipoint	Yes	No	No
Linestring	Yes	No	No
Multilinestring	Yes	No	No
Polygon	Yes	No	Yes
Multipolygon	Yes	No	No
GeometryCollection	Yes	No	No

Example

The following example shows how to use STV_IsValidReason.

Returns a string describing where the polygon is invalid:

```
=> SELECT STV_IsValidReason(ST_GeomFromText('POLYGON((1 3,3 2,1 1,
      3 0,1 0,1 3))'));
      STV_IsValidReason
-----
Ring Self-intersection at or near POINT (1 1)
(1 row)
```

See Also

[ST_IsValid](#)

STV_LineStringPoint

Retrieves the vertices of a linestring or multilinestring. The values returned are points of either GEOMETRY or GEOGRAPHY type depending on the input object's type. GEOMETRY points inherit the SRID of the input object.

STV_LineStringPoint is an analytic function. For more information, see [Analytic Functions](#).

Behavior Type

Immutable

Syntax

```
STV_LineStringPoint( g )  
    OVER( [PARTITION NODES] ) AS
```

Arguments

<i>g</i>	Linestring or multilinestring, value of type GEOMETRY or GEOGRAPHY
----------	--

Returns

GEOMETRY or GEOGRAPHY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)

Point	No	No	No
Multipoint	No	No	No
Linestring	Yes	Yes	Yes
Multilinestring	Yes	Yes	Yes
Polygon	No	No	No
Multipolygon	No	No	No
GeometryCollection	No	No	No

Examples

The following examples show how to use STV_LineStringPoint.

Returns the vertices of the geometry linestring and their SRID:

```
=> SELECT ST_AsText(Point), ST_SRID(Point)
      FROM (SELECT STV_LineStringPoint(
              ST_GeomFromText('MULTILINESTRING((1 2, 2 3, 3 1, 4 2),
                              (10 20, 20 30, 30 10, 40 20))', 4269)) OVER () AS Point) AS foo;
 ST_AsText | ST_SRID
-----+-----
POINT (1 2) | 4269
POINT (2 3) | 4269
POINT (3 1) | 4269
POINT (4 2) | 4269
POINT (10 20) | 4269
POINT (20 30) | 4269
POINT (30 10) | 4269
POINT (40 20) | 4269
(8 rows)
```

Returns the vertices of the geography linestring:

```
=> SELECT ST_AsText(g)
      FROM (SELECT STV_LineStringPoint(
              ST_GeographyFromText('MULTILINESTRING ((42.1 71.0, 41.4 70.0, 41.3 72.9),
                                   (42.99 71.46, 44.47 73.21)', 4269)) OVER () AS g) AS line_geog_points;
 ST_AsText
-----
POINT (42.1 71.0)
POINT (41.4 70.0)
POINT (41.3 72.9)
POINT (42.99 71.46)
POINT (44.47 73.21)
(5 rows)
```

See Also

[STV_PolygonPoint](#)

STV_MemSize

Returns the length of the spatial object in bytes as an INTEGER.

Use this function to determine the optimal column width for your spatial data.

Behavior Type

Immutable

Syntax

```
STV_MemSize( g )
```

Arguments

<i>g</i>	Spatial object, value of type GEOMETRY or GEOGRAPHY
----------	---

Returns

INTEGER

Examples

The following example shows how you can optimize your table by sizing the GEOMETRY or GEOGRAPHY column to the maximum value returned by STV_MemSize:

```
=> CREATE TABLE mem_size_table (id int, geom geometry(800));  
CREATE TABLE  
=> COPY mem_size_table (id, gx filler LONG VARCHAR, geom as ST_GeomFromText(gx)) FROM STDIN DELIMITER  
'|';  
Enter data to be copied followed by a newline.
```

```
End with a backslash and a period on a line by itself.
>>1|POINT(3 5)
>>2|MULTILINESTRING((1 5, 2 4, 5 3, 6 6),(3 5, 3 7))
>>3|MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
>>\.
=> SELECT max(STV_MemSize(geom)) FROM mem_size_table;
  max
-----
  336
(1 row)

=> CREATE TABLE production_table(id int, geom geometry(336));
CREATE TABLE
=> INSERT INTO production_table SELECT * FROM mem_size_table;
OUTPUT
-----
      3
(1 row)
=> DROP mem_size_table;
DROP TABLE
```

STV_NN

Calculates the distance of spatial objects from a reference object and returns (object, distance) pairs in ascending order by distance from the reference object.

Parameters *g1* and *g2* must be both GEOMETRY objects or both GEOGRAPHY objects.

STV_NN is an analytic function. For more information, see [Analytic Functions](#).

Behavior Type

Immutable

Syntax

```
STV_NN( g, ref_obj, k ) OVER()
```

Arguments

<i>g</i>	Spatial object, value of type GEOMETRY or GEOGRAPHY
<i>ref_obj</i>	Reference object, type GEOMETRY or GEOGRAPHY
<i>k</i>	Number of rows to return, type INTEGER

Returns

(Object, distance) pairs, in ascending order by distance. If a parameter is EMPTY or NULL, then 0 rows are returned.

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)
Point	Yes	Yes
Multipoint	Yes	Yes
Linestring	Yes	Yes
Multilinestring	Yes	Yes
Polygon	Yes	Yes
Multipolygon	Yes	Yes
GeometryCollection	Yes	No

Example

The following example shows how to use STV_NN.

Create a table and insert nine GEOGRAPHY points:

```
=> CREATE TABLE points (g geography);
CREATE TABLE
=> COPY points (gx filler LONG VARCHAR, g AS ST_GeographyFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> POINT (21.5 18.4)
>> POINT (21.5 19.2)
>> POINT (21.5 20.7)
>> POINT (22.5 16.4)
>> POINT (22.5 17.15)
>> POINT (22.5 18.33)
>> POINT (23.5 13.68)
>> POINT (23.5 15.9)
>> POINT (23.5 18.4)
>> \.
```

Calculate the distances (in meters) of objects in table `points` from the GEOGRAPHY point (23.5, 20).

Returns the five objects that are closest to that point:

```
=> SELECT ST_AsText(nn), dist FROM (SELECT STV_NN(g,  
  ST_GeographyFromText('POINT(23.5 20)'),5) OVER() AS (nn,dist) FROM points) AS example;  
-----+-----  
ST_AsText | dist  
-----+-----  
POINT (23.5 18.4) | 177912.12757541  
POINT (22.5 18.33) | 213339.210738322  
POINT (21.5 20.7) | 222561.43679943  
POINT (21.5 19.2) | 227604.371833335  
POINT (21.5 18.4) | 275239.416790128  
(5 rows)
```

STV_PolygonPoint

Retrieves the vertices of a polygon as individual points. The values returned are points of either GEOMETRY or GEOGRAPHY type depending on the input object's type. GEOMETRY points inherit the SRID of the input object.

STV_PolygonPoint is an analytic function. For more information, see [Analytic Functions](#).

Behavior Type

Immutable

Syntax

```
STV_PolygonPoint( g )  
  OVER( [PARTITION NODES] ) AS
```

Arguments

<i>g</i>	Polygon, value of type GEOMETRY or GEOGRAPHY
----------	--

Returns

GEOMETRY or GEOGRAPHY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	No	No	No
Multipoint	No	No	No
Linestring	No	No	No
Multilinestring	No	No	No
Polygon	Yes	Yes	Yes
Multipolygon	Yes	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how to use STV_PolygonPoint.

Returns the vertices of the geometry polygon:

```
=> SELECT ST_AsText(g) FROM (SELECT STV_PolygonPoint(ST_GeomFromText('POLYGON((1 2, 2 3, 3 1, 1 2))'))
    OVER (PARTITION NODES) AS g) AS poly_points;
    ST_AsText
-----
POINT (1 2)
POINT (2 3)
POINT (3 1)
POINT (1 2)
(4 rows)
```

Returns the vertices of the geography polygon:

```
=> SELECT ST_AsText(g) FROM (SELECT STV_PolygonPoint(ST_GeographyFromText('
    POLYGON((25.5 28.76, 28.83 29.13, 27.2 30.99, 25.5 28.76))'))
    OVER (PARTITION NODES) AS g) AS poly_points;
    ST_AsText
-----
POINT (25.5 28.76)
POINT (28.83 29.13)
POINT (27.2 30.99)
POINT (25.5 28.76)
```

(4 rows)

See Also

[STV_LineStringPoint](#)

STV_Reverse

Reverses the order of the vertices of a spatial object.

Behavior Type

Immutable

Syntax

```
STV_Reverse( g, [USING PARAMETERS skip_nonreorientable_polygons={true | false} ])
```

Arguments

<i>g</i>	Spatial object, type GEOGRAPHY.
skip_nonreorientable_polygons = { true false }	<p>(Optional) Boolean</p> <p>When set to False, non-orientable polygons generate an error. For example, if you use STV_ForceLHR or STV_Reverse with skip_nonreorientable_polygons set to False, a geography polygon containing a hole generates an error. When set to True, the result returned is the polygon, as passed to the API, without alteration.</p> <p>This argument can help you when you are creating an index from a table containing polygons that cannot be re-oriented.</p> <p>Vertica Place considers these polygons non-</p>

	<p>orientable:</p> <ul style="list-style-type: none"> • Polygons with a hole • Multipolygons • Multipolygons with a hole <p>Default value: False</p>
--	--

Returns

GEOGRAPHY

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (Perfect Sphere)	GEOGRAPHY (WGS84)
Point	No	No	No
Multipoint	No	No	No
Linestring	No	No	No
Multilinestring	No	No	No
Polygon	No	Yes	Yes
Multipolygon	No	Yes	Yes
GeometryCollection	No	No	No

Examples

The following examples show how you can use STV_Reverse.

Reverse vertices of a geography polygon:

```
=> SELECT ST_AsText(STV_Reverse(ST_GeographyFromText('Polygon((1 1, 3 1, 2 2, 1 1))')));
      ST_AsText
-----
```

```
POLYGON ((1 1, 2 2, 3 1, 1 1))  
(1 row)
```

Force the polygon to reverse orientation:

```
=> SELECT ST_AsText(STV_Reverse(ST_GeographyFromText('Polygon((1 1, 2 2, 3 1, 1 1)'))));  
ST_AsText  
-----  
POLYGON ((1 1, 3 1, 2 2, 1 1))  
(1 row)
```

See Also

[STV_ForceLHR](#)

STV_Rename_Index

Renames a spatial index. If the index format is out of date, you cannot rename the index.

A spatial index is created from an input polygon set, which can be the result of a query. Spatial indexes are created in a global name space. Vertica uses a distributed plan whenever the input table or projection is segmented across nodes of the cluster.

The OVER() clause must be empty.

Behavior Type

Immutable

Syntax

```
STV_Rename_Index( USING PARAMETERS  
                  source = 'old_index_name',  
                  dest = 'new_index_name',  
                  overwrite = [ 'true' | 'false' ]  
                )  
OVER ( )
```

Arguments

<code>source = 'old_index_name'</code>	Current name of the spatial index, type VARCHAR.
--	--

<code>dest = 'new_index_name'</code>	New name of the spatial index, type VARCHAR.
<code>overwrite = ['true' 'false']</code>	(Optional) BOOLEAN value that specifies whether to overwrite the index, if an index exists. This parameter cannot be NULL. Default: False

Privileges

Any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

Limitations

- Index names cannot exceed 110 characters.
- The backslash or tab characters are not allowed in index names.

Example

The following example shows how to use STV_Rename_Index.

Rename an index:

```
=> SELECT STV_Rename_Index (  
    USING PARAMETERS  
    source = 'my_polygons',  
    dest = 'US_states',  
    overwrite = 'false'  
    )  
    OVER ();  
rename_index  
-----  
Index renamed  
(1 Row)
```

STV_Refresh_Index

Appends newly added or updated polygons and removes deleted polygons from an existing spatial index.

The OVER() clause must be empty.

Behavior Type

Mutable

Syntax

```
STV_Refresh_Index( gid, g
                  USING PARAMETERS index='index_name'
                               [, skip_nonindexable_polygons={ true | false } ] )
OVER()
  [ AS (type, polygons, srid, min_x, min_y, max_x, max_y, info,
        indexed, appended, updated, deleted) ]
```

Arguments

<i>gid</i>	Name of an integer column that uniquely identifies the polygon. The gid cannot be NULL.
<i>g</i>	Name of a geometry or geography (WGS84) column or expression that contains polygons and multipolygons. Only polygon and multipolygon can be indexed. Other shape types are excluded from the index.

Parameters

<code>index = 'index_name'</code>	Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not allowed in index names.
<code>skip_nonindexable_polygons = { true false }</code>	(Optional) BOOLEAN In rare cases, intricate polygons (for instance, with too high resolution or anomalous spikes) cannot be indexed. These polygons are considered non-indexable. When set to False, non-indexable polygons cause the index creation to fail. When set to True, index creation can succeed by excluding non-indexable polygons from the index.

	<p>To review the polygons that were not able to be indexed, use <code>STV_Describe_Index</code> with the parameter <code>list_polygon</code>.</p> <p>Default: <code>False</code></p>
--	--

Returns

<i>type</i>	Spatial object type of the index.
<i>polygons</i>	Number of polygons indexed.
<i>SRID</i>	Spatial reference system identifier.
<i>min_x, min_y, max_x, max_y</i>	Coordinates of the minimum bounding rectangle (MBR) of the indexed geometries. (<i>min_x, min_y</i>) are the south-west coordinates, and (<i>max_x, max_y</i>) are the north-east coordinates.
<i>info</i>	Lists the number of excluded spatial objects as well as their type that were excluded from the index.
<i>indexed</i>	Number of polygons indexed during the operation.
<i>appended</i>	Number of appended polygons.
<i>updated</i>	Number of updated polygons.
<i>deleted</i>	Number of deleted polygons.

Supported Data Types

Data Type	GEOMETRY	GEOGRAPHY (WGS84)
Point	No	No
Multipoint	No	No
Linestring	No	No
Multilinestring	No	No

Polygon	Yes	Yes
Multipolygon	Yes	No
GeometryCollection	No	No

Privileges

Any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

Limitations

- In rare cases, intricate polygons (such as those with too-high a resolution or anomalous spikes) cannot be indexed. See the parameter `skip_nonindexable_polygons`.
- If you replace a valid polygon in the source table with an invalid polygon, STV_Refresh_Index ignores the invalid polygon. As a result, the polygon originally indexed persists in the index.
- The following geometries cannot be indexed:
 - Non-polygons
 - NULL gid
 - NULL (multi) polygon
 - EMPTY (multi) polygon
 - Invalid (multi) polygon
- The following geographies are excluded from the index:
 - Polygons with holes
 - Polygons crossing the International Date Line
 - Polygons covering the north or south pole
 - Antipodal polygons

Usage Tips

- To cancel an `STV_Refresh_Index` run, use **Ctrl + C**.
- If you use source data not previously associated with the index, then the index will be overwritten.
- If `STV_Refresh_Index` has insufficient memory to process the query, then rebuild the index using `STV_Create_Index`.
- If there are no valid polygons in the `geom` column, `STV_Refresh_Index` reports an error in `vertica.log` and stops the index refresh.
- Ensure that all of the polygons you plan to index are valid polygons. `STV_Create_Index` and `STV_Refresh_Index` do not check polygon validity when building an index.

For more information, see [Ensuring Polygon Validity Before Creating or Refreshing an Index](#).

Examples

The following examples show how to use `STV_Refresh_Index`.

Refresh an index with a single literal argument:

```
=> SELECT STV_Create_Index(1, ST_GeomFromText('POLYGON((0 0,0 15.2,3.9 15.2,3.9 0,0 0))')
      USING PARAMETERS index='my_polygon') OVER();
   type | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |          1 |      0 |      0 |      0 |    3.9 |   15.2 |
(1 row)

=> SELECT STV_Refresh_Index(2, ST_GeomFromText('POLYGON((0 0,0 13.2,3.9 18.2,3.9 0,0 0))')
      USING PARAMETERS index='my_polygon') OVER();
   type | polygons | SRID | min_x | min_y | max_x | max_y | info | indexed | appended | updated |
deleted
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |          1 |      0 |      0 |      0 |    3.9 |   18.2 |      |          1 |          1 |          0 |
1
(1 row)
```

Refresh an index from a table:

```
=> CREATE TABLE polys (gid INT, geom GEOMETRY);
CREATE TABLE
=> COPY polys(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
```

```

End with a backslash and a period on a line by itself.
>> 1|POLYGON((-31 74,8 70,8 50,-36 53,-31 74))
>> 2|POLYGON((5 20,9 30,20 45,36 35,5 20))
>> 3|POLYGON((12 23,9 30,20 45,36 35,37 67,45 80,50 20,12 23))
>> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true)
      OVER() FROM pols;
      type | polygons | SRID | min_x | min_y | max_x | max_y | info
-----+-----+-----+-----+-----+-----+-----+-----
GEOMETRY |         3 |    0 |   -36 |    20 |    50 |    80 |
(1 row)

=> COPY pols(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 6|POLYGON((-32 74,8 70,8 50,-36 53,-32 74))
>> \.
=> SELECT STV_Refresh_Index(gid, geom USING PARAMETERS index='my_polygons_1') OVER() FROM pols;
      type | polygons | SRID | min_x | min_y | max_x | max_y | info | indexed | appended | updated |
deleted
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
-----
GEOMETRY |         4 |    0 |   -36 |    20 |    50 |    80 |      |         1 |         1 |         0 |
0
(1 row)

```

See Also

- [STV_Create_Index](#)
- [STV_Describe_Index](#)
- [STV_Drop_Index](#)
- [STV_Rename_Index](#)
- [Ensuring Polygon Validity Before Creating or Refreshing an Index](#)

STV_SetExportShapefileDirectory

Specifies the directory to export GEOMETRY or GEOGRAPHY data to a shapefile. The validity of the path is not checked, and the path cannot be empty.

Behavior Type

Immutable

Syntax

```
STV_SetExportShapefileDirectory( USING PARAMETERS path='shapefile_path' )
```

Arguments

<code>path = ' shapefile_path '</code>	The path where you want the shapefile exported. For example, '/home/user/temp'.
--	---

Returns

The path of the shapefile export directory.

Privileges

Only a superuser can use this function.

Examples

The following example shows how you can use `STV_SetExportShapefileDirectory` to set the shapefile export directory to `/home/user/temp`:

```
=> SELECT STV_SetExportShapefileDirectory(USING PARAMETERS path = '/home/user/temp');
          STV_SetExportShapefileDirectory
-----
SUCCESS. Set shapefile export directory: [/home/user/temp]
(1 row)
```

STV_ShpSource and STV_ShpParser

These two functions work with a `COPY` command to parse and load the geometries and attributes from a shapefile into a database table and convert them to the `GEOMETRY` data type format. You must use these two functions together. An SRID is required. An empty multipoint or an invalid multipolygon can not be loaded from a shapefile.

Behavior Type

Immutable

Syntax

```
COPY table_name( col2, col3, ..., colN )
  WITH SOURCE STV_ShpSource( file = 'filename'[[, SRID=spatial reference identifier]
                               [, flatten_2d={true | false }]] )
  PARSER STV_ShpParser()
```

Arguments

<i>table_name</i>	Name of the table in which to load the geometry data.
<i>col1</i> , <i>col2</i> , ...	Column names in the table that match the fields in the external file. Run the CREATE TABLE command that STV_ShpCreateTable creates. When you do so, these columns correspond to the second through the second-to-last columns.
file = ' <i>filename</i> '	Fully qualified path of the .dbf, .shp, or .shx file.
SRID= <i>spatial reference identifier</i>	Spatial reference identifier (SRID) associated with the shape file, type INTEGER.
flatten_2d	(Optional) BOOLEAN that excludes 3D or 4D coordinates during COPY commands. True - Excludes geometries with 3D or 4D coordinates before a COPY command. False - Causes the load to fail if a geometry with 3D or 4D coordinate is found. Default: False

Usage Tips

- The COPY command fails if:
 - The shapefile cannot be located or opened.
 - The number of columns or the data types of the columns that STV_ShParser creates do not match the columns in the destination table. Use STV_ShpCreateTable to generate the appropriate CREATE TABLE command.
 - One of the mandatory files is missing or cannot be opened. When opening a shapefile, you must have three files: .dbf, .shp, and .shx.
- If the .shp and .shx files are corrupt, STV_ShpSource returns an error. If the .shp and .shx files are valid, but the .dbf file is corrupt, STV_ShpSource ignores the .dbf file and does not create columns for that data.
- Any rejected records are saved in the /CopyErrorLogs directory under the catalog directory.
- If the .dbf component of a shapefile contains a numeric attribute, this field's values may lose precision when the Vertica Place shapefile loader loads it into a table. The target field is a 64-bit FLOAT column, which can only represent about 15 significant digits; in a .dbf file, Numeric fields can be up to 30 digits.

Example

The following example shows how to use STV_ShpSource and STV_ShParser.

```
=> COPY t1_2010_us_state10 WITH SOURCE
STV_ShpSource(file='/shapefiles/t1_2010_us_state10.shp', SRID=4269) PARSER STV_ShParser();

Rows loaded
-----
          52
```

See Also

- [STV_ShpCreateTable](#)

STV_ShpCreateTable

Returns a CREATE TABLE statement with the columns and types of the attributes found in the specified shapefile.

The column types are sized according to the shapefile metadata. The size of the column is based on the largest geometry found in the shapefile. The first column in the table is `gid`, which is an auto-increment IDENTITY primary key column. The cache value is set to 64 by default. The last column is a GEOMETRY data type for storing the actual geometry data.

Behavior Type

Immutable

Syntax

```
STV_ShpCreateTable(USING PARAMETERS file='filename') OVER()
```

Arguments

<code>file = 'filename'</code>	Fully qualified path of the .dbf, .shp, or .shx file. (The extension is optional.)
--------------------------------	--

Returns

CREATE TABLE statement that matches the specified shapefile

Usage Tips

- STV_ShpCreateTable returns a CREATE TABLE statement; but it does not create the table. Modify the CREATE TABLE statement as needed, and then create the table before loading the shapefile into the table.
- To create a table with characters other than alphanumeric and underscore (`_`) characters, you must specify the table name enclosed in double quotes, such as `"counties%NY"`.

- The name of the table is the same as the name of the shapefile, without the directory name or extension.
- The shapefile must be accessible from the initiator node.
- If the .shp and .shx files are corrupt, STV_ShpCreateTable returns an error. If the .shp and .shx files are valid, but the .dbf file is corrupt, STV_ShpCreateTable ignores the .dbf file and does not create columns for that data.
- All the mandatory files (.dbf, .shp, .shx) must be in the same directory. If not, STV_ShpCreateTable returns an error.
- If the .dbf component of a shapefile contains a Numeric attribute, this field's values may lose precision when the Vertica shapefile loader loads it into a table. The target field is a 64-bit FLOAT column, which can only represent about 15 significant digits. In a .dbf file, numeric fields can be up to 30 digits.

Vertica records all instances of shapefile values that are too long in the `vertica.log` file.

Example

The following example shows how to use STV_ShpCreateTable.

Returns a CREATE TABLE statement:

```
=> SELECT STV_ShpCreateTable
      (USING PARAMETERS file='/shapefiles/tl_2010_us_state10.shp')
      OVER() as create_table_states;
      create_table_states
-----
CREATE TABLE tl_2010_us_state10(
  gid IDENTITY(64) PRIMARY KEY,
  REGION10 VARCHAR(2),
  DIVISION10 VARCHAR(2),
  STATEFP10 VARCHAR(2),
  STATENS10 VARCHAR(8),
  GEOID10 VARCHAR(2),
  STUSPS10 VARCHAR(2),
  NAME10 VARCHAR(100),
  LSAD10 VARCHAR(2),
  MTFCC10 VARCHAR(5),
  FUNCSTAT10 VARCHAR(1),
  ALAND10 INT8,
  AWATER10 INT8,
  INTPTLAT10 VARCHAR(11),
  INTPTLON10 VARCHAR(12),
  geom GEOMETRY(940845)
);
(18 rows)
```

See Also

- [STV_ShpSource](#) and [STV_ShpParser](#)

IP Conversion Functions

IP functions perform conversion, calculation, and manipulation operations on IP, network, and subnet addresses.

INET_ATON

Returns an integer that represents the value of the address in host byte order, given the dotted-quad representation of a network address as a string.

Behavior Type

Immutable

Syntax

```
INET_ATON ( expression )
```

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Notes

The following syntax converts an IPv4 address represented as the string A to an integer I. INET_ATON trims any spaces from the right of A, calls the Linux function [inet_pton](#), and converts the result from network byte order to host byte order using [ntohl](#).

```
=> INET_ATON(VARCHAR A) -> INT8 I
```

If A is NULL, too long, or [inet_pton](#) returns an error, the result is NULL.

Examples

The generated number is always in host byte order. In the following example, the number is calculated as $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$.

```
=> SELECT INET_ATON('209.207.224.40');
inet_aton
-----
3520061480
(1 row)

=> SELECT INET_ATON('1.2.3.4');
inet_aton
-----
16909060
(1 row)

=> SELECT TO_HEX(INET_ATON('1.2.3.4'));
to_hex
-----
1020304
(1 row)
```

See Also

- [INET_NTOA](#)

INET_NTOA

Returns the dotted-quad representation of the address as a VARCHAR, given a network address as an integer in network byte order.

Behavior Type

Immutable

Syntax

```
INET_NTOA ( expression )
```

Parameters

<i>expression</i>	(INTEGER) is the network address to convert.
-------------------	--

Notes

The following syntax converts an IPv4 address represented as integer I to a string A.

INET_NTOA converts I from host byte order to network byte order using [htonl](#), and calls the Linux function [inet_ntop](#).

```
=> INET_NTOA(INT8 I) -> VARCHAR A
```

If I is NULL, greater than 2^{32} or negative, the result is NULL.

Examples

```
=> SELECT INET_NTOA(16909060);
inet_ntoa
-----
1.2.3.4
(1 row)

=> SELECT INET_NTOA(03021962);
inet_ntoa
-----
0.46.28.138
(1 row)
```

See Also

- [INET_ATON](#)

V6_ATON

Converts an IPv6 address represented as a character string to a binary string.

Behavior Type

Immutable

Syntax

V6_ATON (*expression*)

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Notes

The following syntax converts an IPv6 address represented as the character string A to a binary string B.

V6_ATON trims any spaces from the right of A and calls the Linux function [inet_pton](#).

```
=> V6_ATON(VARCHAR A) -> VARBINARY(16) B
```

If A has no colons it is prepended with '::ffff:'. If A is NULL, too long, or if `inet_pton` returns an error, the result is NULL.

Examples

```
=> SELECT V6_ATON('2001:DB8::8:800:200C:417A');
           v6_aton
-----
\001\015\270\000\000\000\000\000\010\010\000 \014Az
(1 row)

=> SELECT V6_ATON('1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\377\377\001\002\003\004
(1 row)
SELECT TO_HEX(V6_ATON('2001:DB8::8:800:200C:417A'));
           to_hex
-----
20010db8000000000000008000200c417a
```

```
(1 row)
=> SELECT V6_ATON('::1.2.3.4');
      v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\001\002\003\004
(1 row)
```

See Also

- [V6_NTOA](#)

V6_NTOA

Converts an IPv6 address represented as varbinary to a character string.

Behavior Type

Immutable

Syntax

V6_NTOA (*expression*)

Parameters

<i>expression</i>	(VARBINARY) is the binary string to convert.
-------------------	--

Notes

The following syntax converts an IPv6 address represented as VARBINARY B to a string A.

V6_NTOA right-pads B to 16 bytes with zeros, if necessary, and calls the Linux function [inet_ntop](#).

```
=> V6_NTOA(VARBINARY B) -> VARCHAR A
```

If B is NULL or longer than 16 bytes, the result is NULL.

Vertica automatically converts the form '::ffff:1.2.3.4' to '1.2.3.4'.

Examples

```
=> SELECT V6_NTOA(' \001\015\270\000\000\000\000\000\010\010\000 \014Az ');
      v6_ntoa
-----
2001:db8::8:800:200c:417a
(1 row)

=> SELECT V6_NTOA(V6_ATON('1.2.3.4'));
      v6_ntoa
-----
1.2.3.4
(1 row)

=> SELECT V6_NTOA(V6_ATON('::1.2.3.4'));
      v6_ntoa
-----
::1.2.3.4
(1 row)
```

See Also

- [V6_ATON](#)

V6_SUBNETA

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a binary or alphanumeric IPv6 address.

Behavior Type

Immutable

Syntax

V6_SUBNETA (*expression1*, *expression2*)

Parameters

<i>expression1</i>	(VARBINARY or VARCHAR) is the string to calculate.
<i>expression2</i>	(INTEGER) is the size of the subnet.

Notes

The following syntax calculates a subnet address in CIDR format from a binary or varchar IPv6 address.

V6_SUBNETA masks a binary IPv6 address B so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared. It then converts to an alphanumeric IPv6 address, appending a slash and N.

```
=> V6_SUBNETA(BINARY B, INT8 N) -> VARCHAR C
```

The following syntax calculates a subnet address in CIDR format from an alphanumeric IPv6 address.

```
=> V6_SUBNETA(VARCHAR A, INT8 N) -> V6_SUBNETA(V6_ATON(A), N) -> VARCHAR C
```

Examples

```
=> SELECT V6_SUBNETA(V6_ATON('2001:db8::8:800:200c:417a'), 28);
 v6_subnetA
-----
2001:db8::/28
(1 row)
```

See Also

- [V6_SUBNETN](#)

V6_SUBNETN

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a varbinary or alphanumeric IPv6 address.

Behavior Type

Immutable

Syntax

`V6_SUBNETN (expression1, expression2)`

Parameters

<i>expression1</i>	(VARBINARY or VARCHAR) is the string to calculate. Notes: <ul style="list-style-type: none">• <code>V6_SUBNETN(<VARBINARY>, <INTEGER>)</code> returns VARBINARY. OR <ul style="list-style-type: none">• <code>V6_SUBNETN(<VARCHAR>, <INTEGER>)</code> returns VARBINARY, after using <code>V6_ATON</code> to convert the <VARCHAR> string to <VARBINARY>.
<i>expression2</i>	(INTEGER) is the size of the subnet.

Notes

The following syntax masks a BINARY IPv6 address **B** so that the **N** left-most bits of **S** form a subnet address, while the remaining right-most bits are cleared.

`V6_SUBNETN` right-pads **B** to 16 bytes with zeros, if necessary and masks **B**, preserving its **N**-bit subnet prefix.

```
=> V6_SUBNETN(VARBINARY B, INT8 N) -> VARBINARY(16) S
```

If **B** is NULL or longer than 16 bytes, or if **N** is not between 0 and 128 inclusive, the result is NULL.

$S = [B]/N$ in [Classless Inter-Domain Routing](#) notation (CIDR notation).

The following syntax masks an alphanumeric IPv6 address **A** so that the **N** leftmost bits form a subnet address, while the remaining rightmost bits are cleared.

```
=> V6_SUBNETN(VARCHAR A, INT8 N) -> V6_SUBNETN(V6_ATON(A), N) -> VARBINARY(16) S
```

Example

This example returns VARBINARY, after using V6_ATON to convert the VARCHAR string to VARBINARY:

```
=> SELECT V6_SUBNETN(V6_ATON('2001:db8::8:800:200c:417a'), 28);
      v6_subnetn
-----
\001\015\260\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
```

See Also

- [V6_ATON](#)
- [V6_SUBNETA](#)

V6_TYPE

Characterizes a binary or alphanumeric IPv6 address B as an integer type.

Behavior Type

Immutable

Syntax

```
V6_TYPE ( expression )
```

Parameters

<i>expression</i>	(VARBINARY or VARCHAR) is the type to convert.
-------------------	--

Notes

V6_TYPE(VARBINARY B) returns INT8 T.

```
=> V6_TYPE(VARCHAR A) -> V6_TYPE(V6_ATON(A)) -> INT8 T
```

The IPv6 types are defined in the Network Working Group's [IP Version 6 Addressing Architecture memo](#).

```
GLOBAL = 0      Global unicast addresses
LINKLOCAL = 1   Link-Local unicast (and Private-Use) addresses
LOOPBACK = 2    Loopback
UNSPECIFIED = 3 Unspecified
MULTICAST = 4   Multicast
```

IPv4-mapped and IPv4-compatible IPv6 addresses are also interpreted, as specified in [IPv4 Global Unicast Address Assignments](#).

- For IPv4, Private-Use is grouped with Link-Local.
- If B is VARBINARY, it is right-padded to 16 bytes with zeros, if necessary.
- If B is NULL or longer than 16 bytes, the result is NULL.

Details

IPv4 (either kind):

0.0.0.0/8	UNSPECIFIED	10.0.0.0/8	LINKLOCAL
127.0.0.0/8	LOOPBACK		
169.254.0.0/16	LINKLOCAL		
172.16.0.0/12	LINKLOCAL		
192.168.0.0/16	LINKLOCAL		
224.0.0.0/4	MULTICAST		
others	GLOBAL		

IPv6:

::0/128	UNSPECIFIED	:::1/128	LOOPBACK
fe80::/10	LINKLOCAL		
ff00::/8	MULTICAST		
others	GLOBAL		

Examples

```
=> SELECT V6_TYPE(V6_ATON('192.168.2.10'));
v6_type
-----
      1
(1 row)

=> SELECT V6_TYPE(V6_ATON('2001:db8::8:800:200c:417a'));
v6_type
-----
      0
(1 row)
```

See Also

- [INET_ATON](#)
- [IP Version 6 Addressing Architecture](#)
- [IPv4 Global Unicast Address Assignments](#)

Machine Learning Functions

The machine learning functions contain algorithms for machine learning and data preparation. Additionally, these functions provide evaluation metrics for models. You can use these evaluation metrics to determine the accuracy of your models.

Vertica machine learning functions do not support temp tables.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Data Preparation Functions

You can use the following functions to pre-process your data:

- [APPLY_NORMALIZE](#) - Use this function to apply normalization parameters saved in a model to specific columns.

- [BALANCE](#) - Use this function to balance your data.
- [DETECT_OUTLIERS](#) - Use this function to remove the outliers from your data.
- [IMPUTE](#) - Imputes missing values in a data set.
- [NORMALIZE](#) - Use this function before running one of the machine learning algorithms on your data.
- [NORMALIZE_FIT](#) - Use this function to compute normalization parameters for specific columns in an input table. The normalization parameters are saved.
- [REVERSE_NORMALIZE](#) - Use this function to reverse the normalization transformation.

Evaluation Functions

You can use the following functions to evaluate your data:

- [APPLY_KMEANS](#) - Assigns each row of an input table to a cluster center from an already-existing k-means model.
- [CONFUSION_MATRIX](#) - Returns a confusion matrix based on both predicted and observed values.
- [GET_MODEL_ATTRIBUTE](#) - Extracts specific model attributes.
- [ERROR_RATE](#) - Returns a table that calculates the rate of incorrect classifications.
- [LIFT_TABLE](#) - Returns a table that compares the predictive quality of a binary classifier model.
- [MSE](#) - Returns a table that displays the mean squared error.
- [ROC](#) - Returns a table that displays the points on a receiver operating characteristic curve.
- [RSQUARED](#) - Returns a table with the R-squared value of the predictions in a linear regression model.
- [SUMMARIZE_MODEL](#) - Returns the summary information of a model.

Prediction Functions

You can use the following functions to apply a model to a table:

- [PREDICT_LINEAR_REG](#) - Applies a linear regression model on an input table.
- [PREDICT_LOGISTIC_REG](#) - Applies a logistic regression model on an input table.
- [PREDICT_NAIVE_BAYES](#) - Applies a Naïve Bayes model on an input table.
- [PREDICT_NAIVE_BAYES_CLASSES](#) - Applies a Naïve Bayes model on an input table and returns the probabilities of classes.
- [PREDICT_RF_CLASSIFIER_CLASSES](#) - Applies a random forest model on an input table and returns the probabilities of classes.
- [PREDICT_RF_CLASSIFIER](#) - Applies a random forest model on an input table.
- [PREDICT_SVM_CLASSIFIER](#) - Applies an SVM classification model on an input table.
- [PREDICT_SVM_REGRESSOR](#) - Applies an SVM regressor model on an input table.

Supervised Learning Functions

You can use the following supervised learning functions to run predictive analytics on a data set:

- [LINEAR_REG](#) - Use this function to model the linear relationship between independent variables and some dependent variable.
- [LOGISTIC_REG](#) - Use this function to model the relationship between independent variables and some dependent variable.
- [NAIVE_BAYES](#) - Use this function to classify your data when features can be assumed independent.
- [RF_CLASSIFIER](#) - Use this function to create an ensemble model of decision trees.
- [SVM_CLASSIFIER](#) - Use this function to assign data to one category or the other.
- [SVM_REGRESSOR](#) - Use this function to predict continuous ordered variables.

Unsupervised Learning Functions

You can use the following unsupervised learning functions to run analytics on a data set:

- [KMEANS](#) - Use this function to cluster data points into k different groups.

APPLY_KMEANS

Assigns each row of an input table or view to a cluster center from an already-existing k-means model.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
APPLY_KMEANS ( col1, col2, ..., coln  
              USING PARAMETERS model_name='name_of_model_created_previously'  
                               [,match_by_pos = 'method'] )
```

Arguments

<code>col1, col2, ..., coln</code>	The columns to use from the input table or view.
------------------------------------	--

Parameters

<code>model_name='name_of_kmeans_model_created_previously'</code>	The name of the k-means model.
<code>match_by_pos= 'method'</code>	<p>(Optional) Valid Values:</p> <ul style="list-style-type: none">• false (default): Input columns will be matched to features in the model based on their names.• true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Privileges

To use `APPLY_KMEANS`, you must either be the `dbadmin`, owner of the model or have `USAGE` privileges. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

The following example shows how you can use the `APPLY_KMEANS` function on an input table. Note that you can mix column names and constants:

```
=> SELECT id, APPLY_KMEANS(Sepal_Length, 2.2, 1.3,
                          Petal_Width USING PARAMETERS
                          model_name='myKmeansModel', match_by_pos='true') FROM iris2;
```

id	APPLY_KMEANS
5	1
10	1
14	1
15	1
21	1
22	1
24	1
25	1
32	1
33	1
34	1
35	1
38	1
39	1
42	1
.	.
.	.
.	.

(60 rows)

The following example shows how you can use the `APPLY_KMEANS` function on an input table, using the `match_by_pos` parameter. Note that providing constants instead of column names works with this parameter:

```
=> SELECT id, APPLY_KMEANS(0,0,0,0 USING PARAMETERS
                          model_name='myKmeansModel', match_by_pos='true')
FROM iris ORDER BY id;
```

id	APPLY_KMEANS
1	1
2	1

```
3 | 1
4 | 1
5 | 1
6 | 1
7 | 1
8 | 1
9 | 1
10 | 1
11 | 1
12 | 1
13 | 1
14 | 1
15 | 1
.
.
.
(150 rows)
```

See Also

- [Clustering Data Using k-means](#)
- [KMEANS](#)

APPLY_NORMALIZE

Applies the normalization parameters saved in a model to a set of specified columns in the input table or view. If any column specified in the function is not contained in the model, the data will pass through unchanged to APPLY_NORMALIZE.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
APPLY_NORMALIZE ( col1,col2 ..., coln
                  USING PARAMETERS model_name='model_name');
```

Arguments

col1, col2 ..., coln

The columns to use from the input table or view.

Parameters

<i>model_name</i>	The name of the model. Model names are case-insensitive. Value must be VARCHAR.
-------------------	---

Privileges

To use `APPLY_NORMALIZE`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the `APPLY_NORMALIZE` function on the `hp` and `cyl` columns in the `mtcars` table, where `hp` is in the normalization model and `cyl` is not in the normalization model.

```
=> SELECT APPLY_NORMALIZE (hp, cyl USING PARAMETERS model_name = 'mtcars_normfit') FROM mtcars;
hp          | cyl
-----+-----
0.62897527217865 | 8
0.681978821754456 | 8
              0 | 4
0.434628963470459 | 8
0.575971722602844 | 8
              1 | 8
0.204946994781494 | 6
0.204946994781494 | 6
0.201413422822952 | 4
0.681978821754456 | 8
0.434628963470459 | 6
0.0494699664413929 | 4
0.0494699664413929 | 4
0.749116599559784 | 8
0.0353356897830963 | 4
0.452296823263168 | 8
0.159010604023933 | 4
0.346289753913879 | 8
0.54063606262207 | 8
0.144876331090927 | 4
0.346289753913879 | 8
0.204946994781494 | 6
0.215547695755959 | 4
0.15194346010685 | 4
```

```
0.250883400440216 | 6
0.250883400440216 | 6
0.452296823263168 | 8
0.452296823263168 | 8
0.434628963470459 | 8
0.137809187173843 | 4
0.045936394482851 | 4
0.187279149889946 | 6
(32 rows)
```

See Also

- [NORMALIZE](#)
- [NORMALIZE_FIT](#)
- [Normalizing Data](#)
- [REVERSE_NORMALIZE](#)

BALANCE

Returns a view with an equal distribution of the input data based on the dependent variable.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Behavior Type

Immutable

Syntax

```
BALANCE ( 'output_view', 'input_relation', 'response_column', 'balance_method'
          [USING PARAMETERS [sampling_ratio=value] ])
```

Arguments

<i>output_view</i>	The name of the View where Vertica saves the balanced data from the chosen <i>input_relation</i> .
--------------------	--

<i>input_relation</i>	The table or view that contains the data the function uses to create a more balanced data set.
<i>response_column</i>	The dependent variable: a VARCHAR or INT column in the <i>input_relation</i> .
<i>balance_method</i>	<p>The imbalanced processing method to use. Selects data from the minority and majority classes.</p> <p>Valid values</p> <ul style="list-style-type: none"> • hybrid_sampling: Performs oversampling and undersampling on different classes so each class is equally represented. • over_sampling: Oversamples on all classes, with the exception of the most majority class, towards the most majority class's cardinality. • under_sampling: Undersamples on all classes, with the exception of the most minority class, towards the most minority class's cardinality. An alias of weighted_sampling. • weighted_sampling: An alias of under_sampling.

Parameters

<i>sampling_ratio=value</i>	<p>The desired ratio between the majority class and the minority class. This value has no effect when used with balance method hybrid_sampling.</p> <p>Default: 1.0</p>
-----------------------------	---

Privileges

To use **BALANCE**, you must either be a superuser or have **CREATE** privileges for the schema of the output view and **SELECT** privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

The following example shows how you can use the BALANCE function:

```
=> CREATE TABLE backyard_bugs (id identity, bug_type int, finder varchar(20));
CREATE TABLE

=> COPY backyard_bugs FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|Ants
>> 1|Beetles
>> 3|Ladybugs
>> 3|Ants
>> 3|Beetles
>> 3|Caterpillars
>> 2|Ladybugs
>> 3|Ants
>> 3|Beetles
>> 1|Ladybugs
>> 3|Ladybugs
>> \.

=> SELECT bug_type, COUNT(bug_type) FROM backyard_bugs GROUP BY bug_type;
 bug_type | COUNT
-----+-----
          2 |      1
          1 |      3
          3 |      7
(3 rows)

=> SELECT BALANCE('backyard_bugs_balanced', 'backyard_bugs', 'bug_type', 'under_sampling');
          BALANCE
-----
Finished in 1 iteration

(1 row)

=> SELECT bug_type, COUNT(bug_type) FROM backyard_bugs_balanced GROUP BY bug_type;
-----+-----
          2 |      1
          1 |      2
          3 |      1
(3 rows)
```

See Also

- [Balancing Imbalanced Data](#)

CONFUSION_MATRIX

Using an input table, returns a confusion matrix based on observed and predicted values.

You cannot pass any inputs to the `OVER()` clause.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
CONFUSION_MATRIX ( target, predictions  
                  [ USING PARAMETERS [num_classes=C] ] )  
                  OVER()
```

Arguments

<i>target</i>	The column in the input table containing the response variable. Must be an integer.
<i>predictions</i>	The column in the input table containing the prediction variables. Must be an integer.

Parameters

<i>num_classes=C</i>	(Optional) The number of classes you want to pass to the function. Must be a positive integer. The result of <code>CONFUSION_MATRIX</code> is a table with $C+1$ columns and C rows.
----------------------	---

Examples

This example shows how you can execute the `CONFUSION_MATRIX` function on an input table named `mtcars`. The response variables appear in the column `obs`, while the prediction variables appear in the column `pred`. Because this problem regards classification, both the response variable values and the prediction variable values are either 0 or 1, indicating binary classification.

In the table returned, all 12 cars with a value of 1 in the `am` column were correctly predicted by `PREDICT_LOGISTIC_REG` as having a value of 1. Out of the 20 cars that had a value of 0 in the `am` column, 19 were correctly predicted to have the value 0. One car was incorrectly classified as having the value 1.

```
=> SELECT CONFUSION_MATRIX(obs, pred USING PARAMETERS num_classes=2) OVER()
      FROM (SELECT am AS obs, PREDICT_LOGISTIC_REG(mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb
            USING PARAMETERS model_name='mtcars_log')::INT AS pred
            FROM mtcars) AS prediction_output;
class | 0 | 1 |
-----+---+---+-----
      0 | 19 | 0 |
      1 | 1 | 12 | Of 32 rows, 32 were used and 0 were ignored
(2 rows)
```

DETECT_OUTLIERS

Returns the outliers in a data set based on the outlier threshold. The output is a view containing the outliers.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Behavior Type

Immutable

Syntax

```
DETECT_OUTLIERS ( 'output_table', 'input_relation', 'input_columns', 'outlier_method'
                  USING PARAMETERS outlier_threshold=value
                  [, exclude_columns='col1, col2, ... coln',]
                  [partition_columns='col1, col2, ... coln'])
```

Arguments

<code>output_table</code>	The name of the table where Vertica saves the outliers from the chosen <code>input_columns</code> .
<code>input_relation</code>	The table or view that contains outlier data.
<code>input_columns</code>	The columns of <code>input_relation</code> to be used for determining outliers. The

	<code>input_columns</code> argument supports the use of a wildcard (*) character in place of column names.
<code>outlier_method</code>	The outlier method to use: Valid Values: <ul style="list-style-type: none"> robust_zscore

Parameters

<code>outlier_threshold=value</code>	(Optional) The value beyond which a data point becomes an outlier. Default value: 3.0
<code>exclude_columns='col1, col2, ... col'</code>	(Optional) The columns from <code>input_relation</code> which you want to exclude from the <code>input_columns</code> argument.
<code>partition_columns='col1, col2, ... coln'</code>	(Optional) A comma-separated list of column names from the <code>input_relation</code> which defines the partitions. The function will detect outliers among each partition separately. The default value is empty.

Privileges

To use `DETECT_OUTLIERS`, you must either be a superuser or have `CREATE` privileges for the schema and `SELECT` privileges for the table.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

The following example shows how you can use the `DETECT_OUTLIERS` function:

```
=> CREATE TABLE baseball_roster (id identity, last_name varchar(30), hr int, avg float);
CREATE TABLE

=> COPY baseball_roster FROM STDIN;
Enter data to be copied followed by a newline.
```

End with a backslash and a period on a line by itself.

```
>> Polo|7|.233
>> Gloss|45|.170
>> Gus|12|.345
>> Gee|1|.125
>> Laus|3|.095
>> Hilltop|16|.222
>> Wicker|78|.333
>> Scooter|0|.121
>> Hank|999999|.8888
>> Popup|35|.378
>> \.
```

```
=> SELECT * FROM baseball_roster;
```

id	last_name	hr	avg
3	Gus	12	0.345
4	Gee	1	0.125
6	Hilltop	16	0.222
10	Popup	35	0.378
1	Polo	7	0.233
7	Wicker	78	0.333
9	Hank	999999	0.8888
2	Gloss	45	0.17
5	Laus	3	0.095
8	Scooter	0	0.121

(10 rows)

```
=> SELECT DETECT_OUTLIERS('baseball_outliers', 'baseball_roster', '*', 'robust_zscore' USING
PARAMETERS
outlier_threshold=3.0, exclude_columns='id');
```

```
DETECT_OUTLIERS
-----
Detected 2 outliers

(1 row)
```

```
=> SELECT * FROM baseball_outliers;
```

id	last_name	hr	avg
7	Wicker	78	0.333
9	Hank	999999	0.8888

(2 rows)

ERROR_RATE

Using an input table, returns a table which calculates the rate of incorrect classifications.

You cannot pass any inputs to the OVER() clause.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
ERROR_RATE ( target, predictions  
            [ USING PARAMETERS [num_classes=C] ] )  
OVER()
```

Arguments

<i>target</i>	The column in the input table containing the response variable. Must be an integer.
<i>predictions</i>	The column in the input table containing the prediction variables. Must be an integer.

Parameters

num_classes= <i>C</i>	(Optional) The number of classes you want to pass to the function. Must be a positive integer. The result of ERROR_RATE is a table with 2 columns and <i>C</i> +1 rows. The +1 row is an additional row for the total error rate across classes.
-----------------------	---

Privileges

To use ERROR_RATE, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can execute the ERROR_RATE function on an input table named `mtcars`. The response variables appear in the column `obs`, while the prediction variables appear in the column `pred`. Because this example is a classification problem, all of the response variable values and the prediction variable values are either 0 or 1, indicating binary classification.

In the table returned by the function, the first column displays the class id column. The second column displays the corresponding error rate for the class id. The third column indicates how many rows were successfully used by the function and whether any rows were ignored.

```
=> SELECT ERROR_RATE(obs, pred::int USING PARAMETERS num_classes=2) OVER()
      FROM (SELECT am AS obs, PREDICT_LOGISTIC_REG (mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb
            USING PARAMETERS model_name='logisticRegModel', type='response') AS pred
            FROM mtcars) AS prediction_output;
class | error_rate | comment
-----+-----+-----
0 | 0 | 
1 | 0.0769230797886848 | 
| 0.03125 | Of 32 rows, 32 were used and 0 were ignored
(3 rows)
```

GET_MODEL_ATTRIBUTE

Extracts either a specific attribute from a model or all attributes from a model. Use this function to view a list of attributes and row counts or view detailed information about a single attribute. The output of GET_MODEL_ATTRIBUTE is a table format where users can select particular columns or rows.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
GET_MODEL_ATTRIBUTE (USING PARAMETERS model_name='model name',
                    [attr_name='value'])
```

Parameters

<i>model_name</i>	The name of the model. Model names are case-insensitive.
<i>attr_name</i>	(Optional) The name of the model attribute. If the value is not specified, the function shows all available attributes. Attribute names are case-sensitive.

Privileges

To use `GET_MODEL_ATTRIBUTE`, you must either be the `dbadmin`, owner of the model or have `USAGE` privileges. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can view a summary of all available attributes for a model.

```
=> SELECT GET_MODEL_ATTRIBUTE (USING PARAMETERS model_name='myLinearRegModel');
attr_name      | attr_fields | #_of_rows
-----+-----+-----
data           | coeffNames, coeff, stdErr, zValue, pValue | 2
regularization | type, lambda | 1
iterations     | iterations | 1
skippedRows    | skippedRows | 1
processedRows  | processedRows | 1
callStr        | callStr | 1
(6 rows)
```

This example shows how you can extract the *data* attribute from the *myLinearRegModel* model.

```
=> SELECT GET_MODEL_ATTRIBUTE (USING PARAMETERS model_name='myLinearRegModel', attr_name='data');
coeffNames | coeff | stdErr | zValue | pValue
-----+-----+-----+-----+-----
Intercept | -1.87401598641074 | 0.160143331525544 | -11.7021169008952 | 7.3592939615234e-26
waiting | 0.0756279479518627 | 0.00221854185633525 | 34.0890336307608 | 8.13028381124448e-100
(2 rows)
```

IMPUTE

Imputes missing values in a data set with either the mean or the mode, based on observed values for a variable. This function supports both numeric and categorical data types.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
IMPUTE( 'output_view', 'input_relation', 'input_columns', 'method'
        [ USING PARAMETERS [exclude_columns='col1, col2, ... coln'],]
```

[partition_columns='col1, col2, ... coln'])

Arguments

<i>output_view</i>	<p>The name of the View where the missing-value imputed rows and rows without missing values are stored.</p> <p>The value must be VARCHAR.</p>
<i>input_relation</i>	<p>The table or view that contains the data for missing-value imputation. The value must be VARCHAR.</p>
<i>input_columns</i>	<p>A comma-separated list of the columns in <i>input_relation</i> containing the values used in missing value imputation. The value must be VARCHAR.</p>
<i>method</i>	<p>The missing-value imputation method to use.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • mean: for numeric missing-value imputation • mode: for categorical missing-value imputation <p>Numeric and categorical input values are supported.</p>

Parameters

<i>exclude_columns=col1, col2, ... coln</i>	<p>(Optional) The columns from <i>input_relation</i> that you want to exclude from the <i>input_columns</i> argument.</p> <p>Default Value: Empty</p>
<i>partition_columns=col1, col2, ... coln</i>	<p>(Optional) A comma-separated list of column names from <i>input_relation</i> for the partition clause.</p> <p>Default Value: Empty</p>

Privileges

To use IMPUTE, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges

needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

These examples show how you can use the IMPUTE function on the `small_input_impute` table.

Execute the IMPUTE function, specifying the mean method:

```
=> SELECT impute('output_view','small_input_impute', 'pid, x1,x2,x3,x4','mean' USING PARAMETERS
exclude_columns='pid');
impute
-----
Finished in 1 iteration
(1 row)
```

Execute the IMPUTE function, specifying the mode method:

```
=> SELECT impute('output_view3','small_input_impute', 'pid, x5,x6','mode' USING PARAMETERS exclude_
columns='pid');
impute
-----
Finished in 1 iteration
(1 row)
```

See Also

[Imputing Missing Values](#)

KMEANS

Executes the k-means algorithm on an input table or view. The result is a model with a list of cluster centers.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
KMEANS ( 'model_name', 'input_relation', 'input_columns', num_clusters
        [ USING PARAMETERS [exclude_columns=['col1, col2, ... coln'],]
```

```
[max_iterations=value,]
[epsilon=value,]
[init_method=method,]
[initial_centers_table=table_name,]
[output_view='output_view',]
[key_columns='key_columns' ])
```

Arguments

<i>model_name</i>	The name of the k-means model. Model names are case insensitive.
<i>input_relation</i>	The table or view that contains the input data for k-means.
<i>input_columns</i>	The columns of <i>input_relation</i> to be used for clustering. The <i>input_columns</i> argument supports the use of wildcard (*) characters in place of column names.
<i>num_clusters</i>	The number of clusters you want to create. This argument represents the <i>k</i> in k-means. Must be an INT and greater than zero.

Parameters

<i>exclude_columns='col1, col2, ... coln'</i>	(Optional) The columns from <i>input_relation</i> that you want to exclude from clustering.
<i>max_iterations=value</i>	(Optional) The maximum number of iterations the algorithm performs. If you set this value to a number lower than the number of iterations needed for convergence, the algorithm may not converge. Default Value: 10
<i>epsilon=value</i>	(Optional) Determines whether the algorithm has converged. If, after an iteration, no component of any cluster center changes more than the value of epsilon, the algorithm has converged. Default Value: 1e-4
<i>init_method=method</i>	(Optional) The method used to find the initial cluster centers. You cannot use this parameter if the <i>initial_centers</i> parameter has a value. Providing

	<p>a value for both <code>init_method</code> and <code>initial_centers_table</code> causes Vertica to return an error.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • random • kmeanspp — kmeans++ algorithm <p>Default Value: kmeanspp</p>
<code>initial_centers_table=table_name</code>	<p>(Optional) The table with the initial cluster centers to use. Supply this value if you know the initial centers you want to use and do not want Vertica to find the initial cluster centers for you.</p> <p>You cannot use this parameter if the <code>init_method</code> parameter has a value. Providing a value for both <code>init_method</code> and <code>initial_centers_table</code> causes Vertica to return an error.</p>
<code>output_view='output_view'</code>	<p>(Optional) The name of the View where you save the assignments of each point to its cluster.</p>
<code>key_columns='key_columns'</code>	<p>(Optional) A comma-separated list of column names from the <code>input_relation</code> which you use to identify each row of the output in the <code>output_view</code>.</p>

Privileges

To use KMEANS, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

The following example shows how you can use the KMEANS function and view the results of the model in the `output_view`.

```
=> SELECT KMEANS('myKmeansModel', 'iris1', '*', 5
                USING PARAMETERS max_iterations=20, output_view='myKmeansView', key_columns='id',
```

```
                exclude_columns='Species, id');
KMEANS
-----
Finished in 12 iterations

(1 row)
```

See Also

- [Clustering Data Using k-means](#)
- [APPLY_KMEANS](#)

LIFT_TABLE

Returns a table that compares the predictive quality of a logistic regression model. This function is also known as a *lift chart*.

You cannot pass any inputs to the `OVER()` clause.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
LIFT_TABLE ( target, probabilities
            [ USING PARAMETERS [num_bins=nBins] ])
OVER()
```

Arguments

<i>target</i>	The column in the input table containing the response variable. Must be an integer.
<i>probabilities</i>	The column in the input table where the observation is of class 1. Must be a float.

Parameters

<code>num_bins=<i>nBins</i></code>	<p>(Optional) Groups rows together, based upon the probability column, for faster processing. You use this parameter to determine the number of different decision boundaries to consider. The parameter partitions the number line from 0 to 1 in <i>nBin</i> points that are equally spaced. It evaluates the table at each of the <i>nBin</i> points. Must be an integer.</p> <p>Default Value: 100</p>
------------------------------------	---

Examples

This example demonstrates how you can execute the `LIFT_TABLE` function on an input table named `mtcars`.

```
=> SELECT LIFT_TABLE(obs, prob USING PARAMETERS num_bins=2) OVER()
       FROM (SELECT am AS obs, PREDICT_LOGISTIC_REG(mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb
                                                    USING PARAMETERS model_name='logisticRegModel',
                                                    type='probability') AS prob
              FROM mtcars) AS prediction_output;
```

decision_boundary	positive_prediction_ratio	lift	comment
1	0	NaN	
0.5	0.40625	2.46153846153846	
0	1	1	Of 32 rows, 32 were used and 0 were ignored (3 rows)

The first column, `decision_boundary`, indicates the cut-off point for whether to classify a response as 0 or 1. For instance, for each row, if `prob` is greater than `decision_boundary`, the response is classified as 1. If `prob` is less than `decision_boundary`, the response is classified as 0.

The second column, `positive_prediction_ratio`, shows the percentage of samples in class 1 that the function classified correctly using the corresponding `decision_boundary` value.

For the third column, `lift`, the function divides the `positive_prediction_ratio` by the percentage of rows correctly classified as class 1.

LINEAR_REG

Executes linear regression on an input table or view. The result is a linear regression model.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
LINEAR_REG ( 'model_name', 'input_relation', 'response_column', 'predictor_columns'  
            [ USING PARAMETERS[exclude_columns='col1, col2, ... coln',]  
              [optimizer='value',]  
              [epsilon=value,]  
              [max_iterations=value,]  
              [regularization= 'value',]  
              [lambda= value] ])
```

Arguments

<i>model_name</i>	The name of the linear regression model. You can use the resulting model for prediction with the PREDICT_LINEAR_REG function. Model names are case insensitive.
<i>input_relation</i>	The table or view that contains the training data for building the model.
<i>response_column</i>	The name of the column in the <i>input_relation</i> that represents the dependent variable, or outcome. For the model to be valid, all values in this column must be of type numeric.
<i>predictor_columns</i>	A comma-separated list of the columns in the <i>input_relation</i> that represent the independent variables for the model. For the model to be valid, all values in this column must be of type numeric. Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, all the columns in <i>input_relation</i> are selected. In this case, the <i>response_column</i> must be explicitly excluded using the list assigned to <i>exclude_columns</i> .

Parameters

<code>exclude_columns=col1, col2, ... coln</code>	<p>(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>predictor_columns</code> argument.</p>
<code>optimizer=value</code>	<p>(Optional) The optimizer method used to train the model.</p> <p>Valid Values:</p> <ul style="list-style-type: none">• BFGS• Newton (default value)
<code>epsilon=value</code>	<p>(Optional) Determines whether the algorithm has reached the specified accuracy result.</p> <p>Default Value: 1e-6</p>
<code>max_iterations=value</code>	<p>(Optional) Determines the maximum number of iterations the algorithm performs before achieving the specified accuracy result.</p> <p>Default Value: 100</p>
<code>regularization=value</code>	<p>(Optional) Determines the method of regularization.</p> <p>Default Value: None</p> <p>Valid Values:</p> <ul style="list-style-type: none">• L2• None
<code>lambda=value</code>	<p>(Optional) The regularization parameter value. The value must be zero or positive.</p> <p>Default Value: 0.0</p>

Privileges

To use `LINEAR_REG`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the `LINEAR_REG` function.

```
=> SELECT LINEAR_REG('myLinearRegModel', 'faithful', 'eruptions', 'waiting'
                    USING PARAMETERS optimizer='BFGS');
          LINEAR_REG
-----
Finished in 10 iterations

(1 row)
```

LOGISTIC_REG

Executes logistic regression on an input table or view. The result is a logistic regression model.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
LOGISTIC_REG ( 'model_name', 'input_relation', 'response_column', 'predictor_columns'
               [ USING PARAMETERS[exclude_columns='col1, col2, ... coln',]
               [optimizer='value',]
               [epsilon=value, ]
               [max_iterations=value]
               [regularization= 'value',]
               [lambda= value] ])
```

Arguments

<code>model_name</code>	The name of the linear regression model. You can use the resulting model for prediction with the PREDICT_LOGISTIC_REG function.
-------------------------	---

	Model names are case insensitive.
<i>input_relation</i>	The table or view that contains the training data for building the model.
<i>response_column</i>	<p>The name of the column in the <code>input_relation</code> that represents the dependent variable, or outcome.</p> <p>For the model to be valid, all values in this column must be INTEGER types with a value of either 0 or 1. The function automatically skips all other values.</p>
<i>predictor_columns</i>	<p>A comma-separated list of the columns in the <code>input_relation</code> that represent the independent variables for the model.</p> <p>For the model to be valid, all values in this column must be of type numeric.</p> <p>Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, all the columns in <code>input_relation</code> are selected. In this case, the <code>response_column</code> must be explicitly excluded using the list assigned to <code>exclude_columns</code>.</p>

Parameters

<code>exclude_columns=col1, col2, ... coln</code>	<p>(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>predictor_columns</code> argument.</p>
<code>optimizer=value</code>	<p>(Optional) The optimizer method to be used to train the model.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • BFGS • Newton (default value)
<code>epsilon=value</code>	<p>(Optional) Determines whether the algorithm has reached the specified accuracy result.</p> <p>Default Value: 0.000001</p>
<code>max_iterations=value</code>	<p>(Optional) Determines the maximum number of iterations the algorithm performs before achieving the</p>

	<p>specified accuracy result.</p> <p>Default Value: 100</p>
<code>regularization=value</code>	<p>(Optional) Determines the method of regularization.</p> <p>Default Value: None</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • L2 • None
<code>lambda=value</code>	<p>(Optional) The regularization parameter value. The value must be zero or positive.</p> <p>Default Value: 0.0</p>

Privileges

To use LOGISTIC_REG, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the LOGISTIC_REG function.

```
=> SELECT LOGISTIC_REG('myLogisticRegModel', 'mtcars', 'am',
                        'mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb'
                        USING PARAMETERS exclude_columns='hp', optimizer='BFGS');
LOGISTIC_REG
-----
Finished in 20 iterations

(1 row)
```

MSE

Returns a table that displays the mean squared error of the prediction and response columns in a linear regression model.

You cannot pass any inputs to the OVER() clause.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
MSE ( target, prediction)  
      OVER()
```

Parameters

<i>target</i>	The response variable for the model. Must be a float.
<i>prediction</i>	The output from the PREDICT_LINEAR_REG function. If that output is saved as a table, the column containing the prediction from the function is used. Must be a float.

Examples

This example shows how you can execute the MSE function on an input table named `faithful_testing`. The response variables appear in the column `obs`, while the prediction variables appear in the column `pred`.

```
=> SELECT MSE(obs, prediction) OVER()  
      FROM (SELECT eruptions AS obs,  
              PREDICT_LINEAR_REG (waiting USING PARAMETERS model_name='linearRegModel') AS  
prediction  
              FROM faithful_testing) AS prediction_output;  
      mse | Comments  
-----+-----  
0.244712410708555 | Of 272 rows, 272 were used and 0 were ignored  
(1 row)
```

NAIVE_BAYES

Executes the Naive Bayes algorithm on an input table or view. The result is a Naive Bayes model.

The following explains how columns are treated based on their data type:

- FLOAT - Values are assumed to follow some Gaussian distribution.
- INTEGER - Values are assumed to belong to one multinomial distribution.
- CHAR/VARCHAR - Values are assumed to follow some categorical distribution.
- BOOLEAN - Values are treated as categorical with two values.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
NAIVE_BAYES ( 'model_name', 'input_relation', 'response_column', 'predictor_col1, predictor_col2, ..., predictor_coln'
              [ USING PARAMETERS[exclude_columns='col1, col2, ..., coln',]
                [alpha=value] ] )
```

Arguments

<i>model_name</i>	The name of the model. Model names are case insensitive.
<i>input_relation</i>	The table or view that contains the training data for building the model.
<i>response_column</i>	The name of the column in the <i>input_relation</i> that represents the dependent variable, or outcome. This column must be discrete labels that represent different class labels.
<i>predictor_columns</i>	A comma-separated list of the columns in the <i>input_relation</i> that represent the independent variables for the model. Supported Data Types: INT, CHAR/VARCHAR, FLOAT, BOOLEAN Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, the function selects all the columns in <i>input_relation</i> . In this case, the <i>response_column</i> must be explicitly excluded, using the list assigned to <i>exclude_columns</i> .

Parameters

<code>exclude_columns=col1, col2, ..., coln</code>	(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>predictor_columns</code> argument.
<code>alpha=value</code>	(Optional) The parameter used to control Laplace smoothing. Specifies use of Laplace smoothing if the event model is categorical, multinomial, or Bernoulli. Default Value: 1.0

Privileges

To use `NAIVE_BAYES`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the `NAIVE_BAYES` function.

```
=> SELECT NAIVE_BAYES('naive_house84_model', 'house84_train', 'party', '*'
                     USING PARAMETERS exclude_columns='party, id');
                     NAIVE_BAYES
-----
Finished. Accepted Rows: 324 Rejected Rows: 0
(1 row)
```

See Also

- [Classifying Data Using Naive Bayes](#)
- [PREDICT_NAIVE_BAYES](#)
- [PREDICT_NAIVE_BAYES_CLASSES](#)

NORMALIZE

Runs a normalization algorithm on an input table or view. The output is a view with the normalized data.

Note: This function differs from `NORMALIZE_FIT`, which creates and stores a model rather than creating a view definition. This could lead to different performance characteristics between the two algorithms.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
NORMALIZE ( 'output_view', 'input_relation', 'input_columns', 'normalization_method'  
           [ USING PARAMETERS [exclude_columns= 'col1, col2, ... coln ' ] ]
```

Arguments

<i>output_view</i>	The name of the View where you save the normalized data from the chosen <code>input_columns</code> .
<i>input_relation</i>	The table or view that contains the data to be normalized.
<i>input_columns</i>	A comma-separated list of the columns in the <code>input_relation</code> containing the values to be normalized. To include all of the columns from the <code>input_relation</code> , specify this argument as a wildcard (*).
<i>normalization_method</i>	The normalization method to use. Valid Values: <ul style="list-style-type: none">• minmax• zscore• robust_zscore

	If infinity values appear in the table, the method ignores those values.
--	--

Parameters

<code>exclude_columns=col1, col2, ... coln</code>	(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>input_columns</code> argument.
---	--

Privileges

To use `NORMALIZE`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

These examples show how you can use the `NORMALIZE` function on the `wt` and `hp` columns in the `mtcars` table.

Execute the `NORMALIZE` function, and specify the `minmax` method:

```
=> SELECT NORMALIZE('mtcars_norm', 'mtcars',
                   'wt, hp', 'minmax');
      NORMALIZE
-----
Finished in 1 iteration

(1 row)
```

Execute the `NORMALIZE` function, and specify the `zscore` method:

```
=> SELECT NORMALIZE('mtcars_normz', 'mtcars',
                   'wt, hp', 'zscore');
      NORMALIZE
-----
Finished in 1 iteration

(1 row)
```

Execute the `NORMALIZE` function, and specify the `robust_zscore` method:

```
=> SELECT NORMALIZE('mtcars_normz', 'mtcars',
                   'wt, hp', 'robust_zscore');
```

```
NORMALIZE
-----
Finished in 1 iteration

(1 row)
```

See Also

- [Normalizing Data](#)

NORMALIZE_FIT

Computes normalization parameters for each of the specified columns in an input table or view. The resulting model stores the normalization parameters. For example, for MinMax normalization, the minimum and maximum value of each column are stored in the model. The generated model will serve as input to the functions [APPLY_NORMALIZE](#) and [REVERSE_NORMALIZE](#).

For robust_zscore, `NORMALIZE_FIT` uses the [APPROXIMATE_MEDIAN \[Aggregate\]](#) function.

This function differs from `NORMALIZE`, which directly outputs a view with normalized results, rather than storing normalization parameters into a model for later operation.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
NORMALIZE_FIT ( 'model_name', 'input_relation', 'input_columns', 'normalization_method'
                [USING PARAMETERS [exclude_columns='col1, col2, ... coln',]
                [output_view='output_view' ] ] )
```

Arguments

<i>model_name</i>	The name of the model. Model names are case-insensitive.
<i>input_relation</i>	The table or view that contains the data to be normalized.
<i>input_columns</i>	A comma-separated list of the columns in <code>input_relation</code> that contains the values to be normalized.

	To include all of the columns from the <code>input_relation</code> , specify this argument as a wildcard (*).
<code>normalization_method</code>	<p>The normalization method to use.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • minmax • zscore • robust_zscore <p>If infinity values, negative infinity values, or NULL values appear in the table, the method ignores those values.</p>

Parameters

<code>exclude_columns='col1, col2, ... coln'</code>	(Optional) The columns from the <code>input_relation</code> that you want to exclude from the <code>input_columns</code> argument.
<code>output_view = output view</code>	(Optional) The name of the View that contains all columns from the <code>input_relation</code> , with the specified <code>input_columns</code> normalized.

Privileges

To use `NORMALIZE_FIT`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

These examples show how you can use the `NORMALIZE_FIT` function on the `wt` and `hp` columns in the `mtcars` table.

Note: If a column contains only one distinct value, when you use the `APPLY_NORMALIZE` function, it returns NaN for values in that column.

Execute the `NORMALIZE_FIT` function, and specify the `minmax` method:

```
=> SELECT NORMALIZE_FIT('mtcars_normfit', 'mtcars', 'wt,hp', 'minmax');  
NORMALIZE_FIT  
-----  
Success  
(1 row)
```

Execute the `NORMALIZE_FIT` function, and specify the `zscore` method:

```
=> SELECT NORMALIZE_FIT('mtcars_normfitz', 'mtcars', 'wt,hp', 'zscore');  
NORMALIZE_FIT  
-----  
Success  
(1 row)
```

Execute the `NORMALIZE_FIT` function, and specify the `robust_zscore` method:

```
=> SELECT NORMALIZE_FIT('mtcars_normfitrz', 'mtcars', 'wt,hp', 'robust_zscore');  
NORMALIZE_FIT  
-----  
Success  
(1 row)
```

See Also

- [APPLY_NORMALIZE](#)
- [NORMALIZE](#)
- [Normalizing Data](#)
- [REVERSE_NORMALIZE](#)

PREDICT_LINEAR_REG

Applies a linear regression model on an input table or view.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_LINEAR_REG ( col1, col2, ... coln  
                    USING PARAMETERS model_name = 'name_of_model'  
                               [, match_by_pos = 'method'] )
```

Arguments

<code>col1, col2, ..., coln</code>	The columns to use from the input table or view.
------------------------------------	--

Parameters

<code>model_name = 'name_of_model'</code>	The name of the linear regression model.
<code>match_by_pos = 'method'</code>	(Optional) Valid Values: <ul style="list-style-type: none">• false (default): Input columns will be matched to features in the model based on their names.• true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type:FLOAT	Returns the predicted value.
------------------------	------------------------------

Examples

The following example shows how you can use the PREDICT_LINEAR_REG function on an input table.

```
=> SELECT PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='linear_reg_faithful')FROM faithful  
ORDER BY id;
```

```
PREDICT_LINEAR_REG
```

```
-----  
4.15403481386324  
2.18505296804024  
3.76023844469864  
2.8151271587036  
4.62659045686076  
2.26381224187316  
4.86286827835952  
4.62659045686076  
1.94877514654148  
4.62659045686076  
2.18505296804024
```

```
.  
. .  
(272 rows)
```

The following example shows how you can use the PREDICT_LINEAR_REG function on an input table, using the match_by_pos parameter. Note that you can replace the column argument with a constant that does not match an input column:

```
=> SELECT PREDICT_LINEAR_REG(55 USING PARAMETERS model_name='linear_reg_faithful',  
                             match_by_pos='true')FROM faithful ORDER BY id;
```

```
PREDICT_LINEAR_REG
```

```
-----  
2.28552115094171  
2.28552115094171  
2.28552115094171  
2.28552115094171  
2.28552115094171  
2.28552115094171  
2.28552115094171
```

```
.  
. .  
(272 rows)
```

PREDICT_LOGISTIC_REG

Applies a logistic regression model on an input table or view.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_LOGISTIC_REG ( col1, col2, ... coln  
                       USING PARAMETERS model_name = 'name_of_model'
```

```
[, type = 'prediction_type',]
[cutoff = probability_cutoff,]
[match_by_pos = 'method'])
```

Arguments

<code>col1, col2, ..., coln</code>	The columns to use from the input table or view.
------------------------------------	--

Parameters

<code>model_name = 'name_of_model'</code>	The name of the logistic regression model.
<code>type = 'prediction_type'</code>	<p>(Optional) Determines the type of prediction for logistic regression.</p> <p>When response is selected the predicted values are 0 or 1. When probability is selected, the output will be the probability of the predicted category to be 1.</p> <p>Valid Values</p> <ul style="list-style-type: none"> • response (Default Value) • probability
<code>cutoff = probability_cutoff</code>	<p>(Optional). Used in conjunction with type. Valid responses are between 0 and 1, exclusive. When the value of type is "response", the returned value of prediction would be 1 if its corresponding probability is bigger than or equal to the value of cutoff; otherwise, it is 0.</p> <p>Default Value: 0.5</p>
<code>match_by_pos= 'method'</code>	<p>(Optional) Valid Values:</p> <ul style="list-style-type: none"> • false (default): Input columns will be matched to features in the model based on their names. • true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: FLOAT	Returns the predicted class or the probability of the predicted class, depending on the response input. The return can be cast to INT or other numeric types when the return is in the probability of the predicted class.
----------------------------	--

Examples

The following example shows how you can use the PREDICT_LOGISTIC_REG function on an input table.

```
=> SELECT car_model,
        PREDICT_LOGISTIC_REG(mpg, cyl, disp, drat, wt, qsec, vs, gear, carb
                               USING PARAMETERS model_name='myLogisticRegModel')
        FROM mtcars;
car_model | PREDICT_LOGISTIC_REG
-----|-----
Camaro Z28 | 0
Fiat 128 | 1
Fiat X1-9 | 1
Ford Pantera L | 1
Merc 450SE | 0
Merc 450SL | 0
Toyota Corona | 0
AMC Javelin | 0
Cadillac Fleetwood | 0
Datsun 710 | 1
Dodge Challenger | 0
Hornet 4 Drive | 0
Lotus Europa | 1
Merc 230 | 0
Merc 280 | 0
Merc 280C | 0
Merc 450SLC | 0
Pontiac Firebird | 0
Porsche 914-2 | 1
Toyota Corolla | 1
Valiant | 0
Chrysler Imperial | 0
Duster 360 | 0
Ferrari Dino | 1
Honda Civic | 1
Hornet Sportabout | 0
Lincoln Continental | 0
Maserati Bora | 1
Mazda RX4 | 1
Mazda RX4 Wag | 1
Merc 240D | 0
Volvo 142E | 1
```

(32 rows)

The following example shows how you can use the PREDICT_LOGISTIC_REG function on an input table, using the match_by_pos parameter. Note that you can any of the column inputs with a constant that does not match an input column. In this example, the mpg column was replaced with the constant 20:

```
=> SELECT car_model,  
         PREDICT_LOGISTIC_REG(20, cyl, disp, drat, wt, qsec, vs, gear, carb  
                               USING PARAMETERS model_name='myLogisticRegModel', match_by_  
pos='true')  
FROM mtcars;
```

car_model	PREDICT_LOGISTIC_REG
AMC Javelin	0
Cadillac Fleetwood	0
Camaro Z28	0
Chrysler Imperial	0
Datsun 710	1
Dodge Challenger	0
Duster 360	0
Ferrari Dino	1
Fiat 128	1
Fiat X1-9	1
Ford Pantera L	1
Honda Civic	1
Hornet 4 Drive	0
Hornet Sportabout	0
Lincoln Continental	0
Lotus Europa	1
Maserati Bora	1
Mazda RX4	1
Mazda RX4 Wag	1
Merc 230	0
Merc 240D	0
Merc 280	0
Merc 280C	0
Merc 450SE	0
Merc 450SL	0
Merc 450SLC	0
Pontiac Firebird	0
Porsche 914-2	1
Toyota Corolla	1
Toyota Corona	0
Valiant	0
Volvo 142E	1

(32 rows)

PREDICT_NAIVE_BAYES

Applies a Naive Bayes model on an input table or view.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_NAIVE_BAYES ( predictor_columns
                     USING PARAMETERS model_name = 'model_name'
                                     [, type = ' { RESPONSE | PROBABILITY } ',]
                                     [class = 'user_input_class', ]
                                     [match_by_pos = 'method' ] )
```

Arguments

<i>predictor_columns</i>	<p>A comma-separated list of the columns in <code>input_relation</code> that represent the independent features for the model.</p> <p>Supports the use of wildcard (*) characters in place of column names.</p>
--------------------------	---

Parameters

<code><i>model_name</i> = '<i>model_name</i>'</code>	<p>The name of the model. Model names are case insensitive.</p>
<code><i>type</i> = '{ RESPONSE PROBABILITY }'</code>	<p>(Optional) Specifies that the function can take the value RESPONSE or PROBABILITY. RESPONSE, using the class result of the prediction. This value uses the highest probability among all possible classes. PROBABILITY defers to the second argument 'class'.</p> <p>Default Value: response</p>
<code><i>class</i> = '<i>user_input_class</i>'</code>	<p>(Optional) Specifies a specific class to use when <code>type</code> is set to PROBABILITY. The predict function returns the probability of belonging to this given class as predicted by the classifier. If <code>class</code> is not specified, its default value is the predicted class -- the highest will be returned. Thus, the predict function returns the probability that the input instance belonging to its predicted class.</p>
<code><i>match_by_pos</i> = '<i>method</i>'</code>	<p>(Optional) Valid Values:</p>

	<ul style="list-style-type: none"> • false (default): Input columns will be matched to features in the model based on their names. • true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.
--	---

Return

Return data type: VARCHAR	Returns the predicted class or the probability of the predicted class, depending on the response input. The return can be cast to INT or other numeric types when the return is in the probability of the predicted class.
------------------------------	--

Examples

This example shows how you can use the PREDICT_NAIVE_BAYES function.

```
=> SELECT party, PREDICT_NAIVE_BAYES (vote1, vote2, vote3
                                     USING PARAMETERS model_name = 'naive_house84_model',
                                     type = 'response')
       AS Predicted_Party
FROM house84_test;
```

party	Predicted_Party
democrat	democrat
democrat	democrat
democrat	democrat
republican	republican
democrat	democrat
republican	republican
democrat	democrat
democrat	democrat
democrat	democrat
democrat	republican
republican	republican
democrat	democrat
republican	republican

.
.

(99 rows)

See Also

- [Classifying Data Using Naive Bayes](#)
- [NAIVE_BAYES](#)
- [PREDICT_NAIVE_BAYES_CLASSES](#)

PREDICT_NAIVE_BAYES_CLASSES

Applies a Naive Bayes model on an input table or view and returns the probabilities of classes.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_NAIVE_BAYES_CLASSES ( predictor_columns
                               USING PARAMETERS model_name = 'model_name'
                                               [,key_columns = 'key_columns',]
                                               [exclude_columns = 'col1, col2, ..., coln',]
                                               [classes = 'class1, class2, ..., classn', ]
                                               [match_by_pos = 'method' ] )
                               OVER()
                               AS (key_columns, Predicted, Probability, class1, class2, ..., classn)
```

Arguments

<i>predictor_columns</i>	A comma-separated list of the columns in <i>input_relation</i> that represent the independent variables for the model. Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, all the columns in <i>input_relation</i> are selected.
--------------------------	---

Parameters

<i>model_name</i> = ' <i>model_name</i> '	The name of the model. Model names are case
---	---

	insensitive.
<code>key_columns = 'key_columns'</code>	(Optional) A comma-separated list of column names from the <code>input_relation</code> which you use to identify each row of the output.
<code>exclude_columns = 'col1, col2, ..., coln'</code>	(Optional) The columns from <code>predictor_columns</code> that you want to exclude. This parameter is useful when using the wildcard (*) in the <code>predictor_columns</code> .
<code>classes = 'class1, class2, ..., classn'</code>	(Optional) Class labels in the model. The probability of belonging to this given class as predicted by the classifier. The values are case sensitive.
<code>match_by_pos= 'method'</code>	<p>(Optional) Valid Values:</p> <ul style="list-style-type: none"> • false (default): Input columns will be matched to features in the model based on their names. • true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: One VARCHAR column and multiple FLOAT columns	The VARCHAR column is named <code>predicted</code> and contains the class label with the highest probability. The first FLOAT column is named <code>probability</code> and contains the probability for the class specified in the <code>predicted</code>
---	---

	column. The other FLOAT columns contain each class listed in the classes column.
--	--

Examples

This example shows how you can use the PREDICT_NAIVE_BAYES_CLASSES function.

```
=> SELECT PREDICT_NAIVE_BAYES_CLASSES (id, vote1, vote2
                                     USING PARAMETERS model_name = 'naive_house84_model',
                                                         key_columns = 'id',
                                                         exclude_columns = 'id',
                                                         classes = 'democrat, republican',
                                                         match_by_pos = 'false')
       OVER() FROM house84_test;
 id | Predicted | Probability | democrat | republican
-----+-----+-----+-----+-----
 21 | democrat  | 0.775473383353576 | 0.775473383353576 | 0.224526616646424
 28 | democrat  | 0.775473383353576 | 0.775473383353576 | 0.224526616646424
 83 | republican | 0.592510497724379 | 0.407489502275621 | 0.592510497724379
102 | democrat  | 0.779889432167111 | 0.779889432167111 | 0.220110567832889
107 | republican | 0.598662714551597 | 0.401337285448403 | 0.598662714551597
125 | republican | 0.598662714551597 | 0.401337285448403 | 0.598662714551597
132 | republican | 0.592510497724379 | 0.407489502275621 | 0.592510497724379
136 | republican | 0.592510497724379 | 0.407489502275621 | 0.592510497724379
155 | republican | 0.598662714551597 | 0.401337285448403 | 0.598662714551597
174 | republican | 0.592510497724379 | 0.407489502275621 | 0.592510497724379
.
.
.
(1 row)
```

See Also

- [Classifying Data Using Naive Bayes](#)
- [PREDICT_NAIVE_BAYES](#)
- [NAIVE_BAYES](#)

PREDICT_RF_CLASSIFIER_CLASSES

Applies a random forest model on an input table or view and returns the probabilities of classes. The predicted class is selected only based on the popular vote of the decision trees in the forest. Therefore, in special cases the calculated probability of the predicted class may not be the highest.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_RF_CLASSIFIER_CLASSES ( col1, col2, ... coln
                               USING PARAMETERS model_name = 'name_of_model'
                               [,key_columns = 'key_columns',]
                               [exclude_columns = 'col1, col2, ..., coln',]
                               [classes = 'class1, class2, ..., classn', ]
                               [match_by_pos = 'method' ] )
                               OVER([window-partition-clause])
```

Arguments

<i>col1, col2, ..., coln</i>	The columns to use from the input table or view.
------------------------------	--

Parameters

<code>model_name = 'name_of_model'</code>	The name of the random forest model. Model names are case-insensitive.
<code>key_columns = 'key_columns'</code>	(Optional) A comma-separated list of column names from <code>input_relation</code> which you use to identify each row of the output.
<code>exclude_columns = 'col1, col2, ..., coln'</code>	(Optional) The columns from <code>predictor_columns</code> that you want to exclude. This parameter is useful when using the wildcard (*) in the <code>predictor_columns</code> .
<code>classes = 'class1, class2, ..., classn'</code>	(Optional) Class labels in the model. The probability

	of belonging to this given class as predicted by the classifier. The values are case sensitive.
<code>match_by_pos= 'method'</code>	<p>(Optional) Valid Values:</p> <ul style="list-style-type: none"> • false (default): Input columns will be matched to features in the model based on their names. • true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: One VARCHAR column and multiple FLOAT columns	The predicted column, of type VARCHAR, contains the class label with the highest vote (popular vote). The first FLOAT column is named probability and contains the probability for the class reported in the predicted column. The other FLOAT columns contain the probability of each class specified in the <code>classes</code> input parameter.
Key columns	Columns with the same value and data type as the matching input columns that are specified in the <code>key_columns</code> input parameter.

Examples

This example shows how you can use the `PREDICT_RF_CLASSIFIER_CLASSES` function.

```
=> SELECT PREDICT_RF_CLASSIFIER_CLASSES(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
      USING PARAMETERS model_name='myRFModel') OVER () FROM iris;
```

predicted	probability
setosa	1
setosa	0.99
setosa	1
setosa	1
setosa	1
setosa	0.97
setosa	1
setosa	0.99

```
.  
. .  
. .  
(150 rows)
```

This example shows how you can use the `PREDICT_RF_CLASSIFIER_CLASSES` function , using the `match_by_pos` parameter:

```
=> SELECT PREDICT_RF_CLASSIFIER_CLASSES(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width  
      USING PARAMETERS model_name='myRFModel', match_by_pos='true') OVER () FROM  
iris;  
predicted | probability  
-----+-----  
setosa    |           1  
. . .  
(150 rows)
```

See Also

- [Classifying Data Using Random Forest](#)
- [RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER](#)
- [SUMMARIZE_MODEL](#)

PREDICT_RF_CLASSIFIER

Applies a random forest model on an input table or view. The predicted class is selected only based on the popular vote of the decision trees in the forest. Therefore, in special cases the calculated probability of the predicted class may not be the highest.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_RF_CLASSIFIER ( col1, col2, ... coln
                      USING PARAMETERS model_name = 'name_of_model'
                                     [, type= ' { RESPONSE | PROBABILITY } ',]
                                     [class= 'user_input_class',]
                                     [match_by_pos = 'method'])
```

Arguments

<i>col1, col2, ..., coln</i>	The columns to use from the input table or view.
------------------------------	--

Parameters

<code>model_name = 'name_of_model'</code>	The name of the random forest model. Model names are case-insensitive.
<code>type = 'method'</code>	<p>(Optional) Determines the type of prediction for random forest.</p> <p>When response is selected the result of the prediction is the class with the highest probability among all possible classes. When probability is selected, the 'class' parameter is considered.</p> <p>Valid Values</p> <ul style="list-style-type: none"> • response (Default Value) • probability
<code>class = 'user_input_class'</code>	(Optional) Specifies a specific class to use when type is set to PROBABILITY. The predict function returns the probability of the specified class to be the response. If class is not specified, its default value is the predicted class -- the one with popular vote. Thus, the predict function returns the probability that the input instance belonging to its predicted class.

	Default Value: Auto
match_by_pos= 'method'	<p>(Optional) Valid Values:</p> <ul style="list-style-type: none"> • false (default): Input columns will be matched to features in the model based on their names. • true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: VARCHAR	The predict function returns the predicted class (based on popular votes) or probability of a class (depending on the value of the optional input parameters type and class) for each input instance.
------------------------------	---

Examples

This example shows how you can use the PREDICT_RF_CLASSIFIER function.

```
=> SELECT PREDICT_RF_CLASSIFIER (Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
                                USING PARAMETERS model_name='myRFModel') FROM iris;
PREDICT_RF_CLASSIFIER
-----
setosa
setosa
setosa
.
.
.
versicolor
versicolor
versicolor
.
.
.
virginica
virginica
virginica
.
.
.
(150 rows)
```

This example shows how you can use the PREDICT_RF_CLASSIFIER function, using the match_by_pos parameter:

```
=> SELECT PREDICT_RF_CLASSIFIER (Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
                                USING PARAMETERS model_name='myRFModel', match_by_pos='true') FROM
iris;
PREDICT_RF_CLASSIFIER
-----
setosa
setosa
setosa
.
.
.
versicolor
versicolor
versicolor
.
.
.
virginica
virginica
virginica
.
.
.
(150 rows)
```

See Also

- [Classifying Data Using Random Forest](#)
- [RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER_CLASSES](#)
- [SUMMARIZE_MODEL](#)

PREDICT_SVM_CLASSIFIER

Applies an SVM model on an input table or view.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_SVM_CLASSIFIER (input_columns
                        USING PARAMETERS model_name='model_name'
                        [, match_by_pos = 'method'])
```

Arguments

<i>input_columns</i>	A comma-separated list of the columns to be used for prediction.
----------------------	--

Parameters

<i>model_name</i> ='model_name'	The name of the model. Model names are case-insensitive.
<i>match_by_pos</i> = 'method'	(Optional) Valid Values: <ul style="list-style-type: none">false (default): Input columns will be matched to features in the model based on their names.true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: FLOAT	Returns the predicted value.
-------------------------	------------------------------

Examples

This example shows how you can use the PREDICT_SVM_CLASSIFIER function on the mtcars table:

```
=> SELECT PREDICT_SVM_CLASSIFIER (mpg,cyl,disp,wt,qsec,vs,gear,carb USING PARAMETERS model_
name='mySvmClassModel') FROM mtcars;
PREDICT_SVM_CLASSIFIER
```

```
-----  
0  
0  
1  
0  
0  
1  
1  
1  
1  
0  
0  
1  
0  
0  
0  
0  
0  
0  
0  
1  
1  
0  
0  
1  
1  
1  
1  
1  
0  
0  
0  
  
(32 rows)
```

This example shows how you can use the `PREDICT_SVM_CLASSIFIER` function on the `mtcars` table, using the `match_by_pos` parameter. Note that you can any of the column inputs with a constant that does not match an input column. In this example, the `mpg` column was replaced with the constant 40:

```
=> SELECT PREDICT_SVM_CLASSIFIER (40,cyl,disp,wt,qsec,vs,gear,carb USING PARAMETERS model_  
name='mySvmClassModel',  
                                match_by_pos ='true') FROM mtcars;  
  
PREDICT_SVM_CLASSIFIER  
-----  
0  
0  
0  
0  
1  
0  
0  
1  
1  
1  
1
```

```
1
1
0
0
0
1
1
1
1
0
0
0
0
0
0
0
0
0
0
1
1
0
0
1
(32 rows)
```

See Also

- [Classifying Data Using SVM \(Support Vector Machine\)](#)
- [SVM \(Support Vector Machine\) for Classification](#)
- [SVM_CLASSIFIER](#)
- [SUMMARIZE_MODEL](#)

PREDICT_SVM_REGRESSOR

Applies an SVM model on an input table or view.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
PREDICT_SVM_REGRESSOR(input_columns
    USING PARAMETERS model_name='model_name'
    [, match_by_pos = 'method'])
```

Arguments

<code>input_columns</code>	A comma-separated list of the columns to be used for prediction.
----------------------------	--

Parameters

<code>model_name='model_name'</code>	The name of the model. Model names are case-insensitive.
<code>match_by_pos= 'method'</code>	(Optional) Valid Values: <ul style="list-style-type: none">• false (default): Input columns will be matched to features in the model based on their names.• true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: FLOAT	Returns the predicted value.
-------------------------	------------------------------

Examples

This example shows how you can use the `PREDICT_SVM_REGRESSOR` function on the `faithful` table:

```
=> SELECT PREDICT_SVM_REGRESSOR(waiting USING PARAMETERS model_name='mySvmRegModel')
       FROM faithful ORDER BY id;
PREDICT_SVM_REGRESSOR
-----
4.06488248694445
2.30392277646291
3.71269054484815
2.867429883817
4.48751281746003
2.37436116488217
4.69882798271781
4.48751281746003
2.09260761120512
```

```
.  
. .  
. .  
(272 rows)
```

This example shows how you can use the `PREDICT_SVM_REGRESSOR` function on the `faithful` table, using the `match_by_pos` parameter. Note that you can any of the column inputs with a constant that does not match an input column. In this example, the waiting column was replaced with the constant 40:

```
=> SELECT PREDICT_SVM_REGRESSOR(40 USING PARAMETERS model_name='mySvmRegModel', match_by_pos =  
'true')  
      FROM faithful ORDER BY id;  
PREDICT_SVM_REGRESSOR  
-----  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
1.31778533859324  
.  
. .  
. .  
(272 rows)
```

See Also

- [Building an SVM for Regression Model](#)
- [SVM \(Support Vector Machine\) for Regression](#)
- [SVM_REGRESSOR](#)
- [SUMMARIZE_MODEL](#)

REVERSE_NORMALIZE

Reverses the normalization transformation on normalized data, thereby de-normalizing the normalized data. If normalized data is used as the input, the original data is returned. If you specify a column that is not in the specified model, `REVERSE_NORMALIZE` returns that column unchanged.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
REVERSE_NORMALIZE ( col1, col2 ..., coln  
                    USING PARAMETERS model_name='model_name');
```

Arguments

<code>col1, col2 ..., coln</code>	The columns to use from the input table or view.
-----------------------------------	--

Parameters

<code>model_name</code>	The name of the model. Model names are case-insensitive. Value must be VARCHAR.
-------------------------	---

Privileges

To use REVERSE_NORMALIZE, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the REVERSE_NORMALIZE function on the hp and cyl columns in the mtcars table, where hp is in the normalization model and cyl is not in the normalization model.

```
=> SELECT REVERSE_NORMALIZE (hp, cyl USING PARAMETERS model_name = 'mtcars_normfit') FROM mtcars;  
hp   | cyl  
-----+-----  
42502 | 8  
58067 | 8  
26371 | 4  
42502 | 8  
31182 | 6
```

```
32031 | 4
26937 | 4
34861 | 6
34861 | 6
50992 | 8
50992 | 8
49577 | 8
25805 | 4
18447 | 4
29767 | 6
65142 | 8
69387 | 8
14768 | 4
49577 | 8
60897 | 8
94857 | 8
31182 | 6
31182 | 6
30899 | 4
69387 | 8
49577 | 6
18730 | 4
18730 | 4
74764 | 8
17598 | 4
50992 | 8
27503 | 4
```

(32 rows)

See Also

- [APPLY_NORMALIZE](#)
- [NORMALIZE](#)
- [NORMALIZE_FIT](#)
- [Normalizing Data](#)

RF_CLASSIFIER

Trains a random forest model for classification on an input table or view.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
RF_CLASSIFIER ( 'model_name', 'input_relation', 'response_column',
                'predictor_col1, predictor_col2, ..., predictor_coln'

                [USING PARAMETERS [exclude_columns= 'col1, col2, ..., coln',]
                [ntree= value,]
                [mtry= value,]
                [sampling_size= value,]
                [max_depth= value,]
                [max_breadth= value,]
                [min_leaf_size= value,]
                [min_info_gain= value,]
                [nbins= value] ])
```

Arguments

<i>model_name</i>	The name of the model stored as a result of the training. Model names are case insensitive.
<i>input_relation</i>	The table or view that contains the training samples.
<i>response_column</i>	The name of the column in <i>input_relation</i> that represents the dependent variable. This column must be of data type CHAR or VARCHAR.
<i>predictor_columns</i>	A comma-separated list of the columns in the <i>input_relation</i> that represent the independent variables for the model. These columns must be of CHAR, VARCHAR, BOOLEAN, INT and FLOAT data types. CHAR, VARCHAR and BOOLEAN are treated as categorical data types. All other data types are treated as numeric data types.

Parameters

<i>exclude_columns='col1, col2, ..., coln'</i>	(Optional) The columns from <i>input_relation</i> that you want to exclude from the <i>predictor_columns</i> argument.
<i>ntree=value</i>	(Optional) A positive integer number that indicates the number of trees in the forest. Default Value: 20

	Valid Range: (0 to 1000]
<code>mtry=value</code>	<p>(Optional) A positive integer number that indicates the number of features to be considered at the split of a tree node.</p> <p>Default Value: When no value is specified for <code>mtry</code>, its default value is the square root of the total number of predictors.</p> <p>Valid Range: A positive integer number, smaller than or equal to the number of predictors.</p>
<code>sampling_size=value</code>	<p>(Optional) A number that indicates what portion of the input data set will randomly be picked for training each tree</p> <p>Default Value: 0.632</p> <p>Valid Range:(0.0,1.0]</p>
<code>max_depth=value</code>	<p>(Optional) A positive integer number that specifies the maximum depth for growing each tree.</p> <p>Default Value: 5</p> <p>Valid Range: [1 to 100]</p>
<code>max_breadth=value</code>	<p>(Optional) A positive integer number that specifies the maximum number of leaf nodes a tree in the forest can have.</p> <p>Default Value: 32</p> <p>Valid Range: [1 to 1e9]</p>
<code>min_leaf_size=value</code>	<p>(Optional) A positive integer number that specifies the minimum samples each branch must have after splitting a node. A split that causes fewer remaining samples will be discarded.</p> <p>Default Value: 1</p> <p>Valid Range: [1 to 1e6]</p>
<code>min_info_gain=value</code>	<p>(Optional) A non-negative number. Any split with information gain less than this threshold will be discarded.</p>

	Default Value: 0.0 Valid Range: [0.0 to 1.0]
<code>nbins=<i>value</i></code>	(Optional) A positive integer number that indicates the number of bins to use for continuous features. Default Value: 32 Valid Range: [2 to 1000]

Privileges

To use `RF_CLASSIFIER`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the `RF_CLASSIFIER` function.

```
=> SELECT RF_CLASSIFIER ('myRFModel', 'iris', 'Species', 'Sepal_Length, Sepal_Width, Petal_Length,
Petal_Width'
USING PARAMETERS ntree=100, sampling_size=0.3);

RF_CLASSIFIER
-----
The random forest is trained
(1 row)
```

See Also

- [Classifying Data Using Random Forest](#)
- [PREDICT_RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER_CLASSES](#)
- [SUMMARIZE_MODEL](#)

ROC

Returns a table that displays the points on a receiver operating characteristic curve. The ROC function tells you the accuracy of a classification model as you raise the discrimination threshold for the model.

You cannot pass any inputs to the `OVER()` clause.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
ROC ( target, probabilities  
      [USING PARAMETERS [num_bins=nBins] ] )  
OVER()
```

Arguments

<i>target</i>	The column in the input table containing the response variable. Must be an integer. Greater values result in more precise approximations of the AUC.
<i>probabilities</i>	The column in the input table where the observation is of class 1. Must be a float.

Parameters

<i>num_bins</i> = <i>nBins</i>	(Optional) Groups rows together, based upon the probability column, for faster processing. You use this parameter to determine the number of different decision boundaries to consider. The parameter partitions the number line from 0 to 1 in <i>nBin</i> points that are equally spaced. It evaluates the table at each of the <i>nBin</i> points. Must be an integer. Default Value: 100
--------------------------------	--

Examples

This example show how you can execute the ROC function on an input table named `mtcars`. The response variables appear in the column `obs`, while the prediction variables appear in the column `pred`.

```
=> SELECT ROC(obs, prob USING PARAMETERS num_bins=2) OVER()
      FROM (SELECT am AS obs,
                PREDICT_LOGISTIC_REG (mpg, cyl, disp, hp, drat, wt,
                                     qsec, vs, gear, carb
                                     USING PARAMETERS model_name='logisticRegModel',
                                                         type='probability') AS prob
                FROM mtcars) AS prediction_output;
```

decision_boundary	false_positive_rate	true_positive_rate	comment
0	1	1	
0.5	0	1	
1	0	0	Of 32 rows, 32 were used and 0 were

ignored
(3 rows)

The first column, `decision_boundary`, indicates the cut-off point for whether to classify a response as 0 or 1. For instance, in each row, if `prob` is greater than `decision_boundary`, the response is classified as 1. If `prob` is less than `decision_boundary`, the response is classified as 0.

The second column, `false_positive_rate`, shows the percentage of false positives (when 0 is classified as 1) in the corresponding `decision_boundary`.

The third column, `true_positive_rate`, shows the percentage of rows that were classified as 1 and also belong to class 1.

RSQUARED

Returns a table with the R-squared value of the predictions in a linear regression model.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
RSQUARED ( target, prediction )
         OVER()
```

Important: You cannot pass any inputs to the OVER() clause.

Parameters

<i>target</i>	The response variable for the model. Must be a float.
<i>prediction</i>	The output from the PREDICT_LINEAR_REG function. If that output is saved as a table, the column containing the prediction from the function is used. Must be a float.

Examples

This example shows how you can execute the RSQUARED function on an input table named `faithful_testing`. The response variables appear in the column, `obs`, while the prediction variables appear in the column, `pred`.

```
=> SELECT RSQUARED(obs, prediction) OVER()  
      FROM (SELECT eruptions AS obs,  
              PREDICT_LINEAR_REG (waiting  
                                USING PARAMETERS model_name='linear_reg_faithful') AS  
prediction  
      FROM faithful_testing) AS prediction_output;  
      rsq      |      comment  
-----+-----  
0.819686091991332 | Of 162 rows, 162 were used and 0 were ignored  
(1 row)
```

SUMMARIZE_MODEL

Returns the summary information of a model.

Note: If you are using Vertica 8.1.0, you must include the owner parameter. For Vertica 8.1.1 and later, the owner parameter has been deprecated.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
SUMMARIZE_MODEL ( 'model_name' [, 'owner' ] )
```

Parameters

<i>model_name</i>	The name of the model you want to summarize. Model names are case insensitive.
<i>owner</i>	(Optional) The name of the user who created the model.

Privileges

Any user can see summary information about any model in the database.

Examples

This example shows how you can view the summary of a linear regression model.

```
=> SELECT SUMMARIZE_MODEL('linear_reg_faithful');
-[ RECORD 1 ]-----+-----
SUMMARIZE_MODEL | coeff names : {Intercept, waiting}
coefficients: {-2.067947819, 0.07875927383}
std_err:      {0.21063, 0.0029203}
t_value:      {-9.8178, 26.969}
p_value:      {4.2157e-18, 2.1094e-61}
Number of iterations: 6, Number of skipped samples: 0, Number of processed samples: 162
Call: linearReg(model_name=linear_reg_faithful, input_table=faithful_training, response_
column=eruptions, predictor_columns=waiting,
exclude_columns=, optimizer=bfgs, epsilon=0.0001, max_iterations=50)
```

SVM_CLASSIFIER

Trains the SVM model on an input table or view. You can view the model using [SUMMARIZE_MODEL](#).

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
SVM_CLASSIFIER ( 'model_name', 'input_relation', 'response_column', 'predictor_columns'
                [USING PARAMETERS [exclude_columns='col1, col2, ... coln',]
```

```
[C='value',]  
[epsilon='value',]  
[max_iterations='value' ] )
```

Arguments

<i>model_name</i>	The name of the model. Model names are case-insensitive.
<i>input_relation</i>	The table or view that contains the training data.
<i>response_column</i>	<p>The name of the column in <i>input_relation</i> that represents the dependent variable, or outcome. The value must be 0 or 1.</p> <p>Valid Types:</p> <ul style="list-style-type: none"> • FLOAT • INT • NUMERIC
<i>predictor_columns</i>	<p>A comma-separated list of the columns in <i>input_relation</i> that represent the independent variables for the model.</p> <p>Valid Types:</p> <ul style="list-style-type: none"> • INT • FLOAT • NUMERIC <p>If the column name contains special characters, it must use double quotes.</p>

Parameters

<code>exclude_columns='col1, col2, ... coln'</code>	<p>(Optional) The columns from <i>input_relation</i> that you want to exclude from the <i>input_columns</i> argument.</p> <p>Default Value: Empty</p>
<code>C='value'</code>	(Optional) Sets the weight for misclassification cost.

	The algorithm minimizes the regularization cost and the misclassification cost. Default Value: 1.0
epsilon= <i>value</i>	(Optional) Used to control accuracy. Default Value: 1e-3
max_iterations= <i>value</i>	(Optional) Determines the maximum number of iterations that the algorithm performs before achieving the specified accuracy result. Default Value: 100

Privileges

To use SVM_CLASSIFIER, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the SVM_CLASSIFIER function on the mtcars table:

```
=> SELECT SVM_CLASSIFIER('mySvmClassModel', 'mtcars', 'am',  
  'mpg,cyl,disp,hp,drat,wt,qsec,vs,gear,carb'  
  USING PARAMETERS exclude_columns = 'hp,drat');  
  
SVM_CLASSIFIER  
-----  
Finished in 15 iterations.  
Accepted Rows: 32 Rejected Rows: 0  
(1 row)
```

See Also

- [Classifying Data Using SVM \(Support Vector Machine\)](#)
- [SVM \(Support Vector Machine\) for Classification](#)

- [PREDICT_SVM_CLASSIFIER](#)
- [SUMMARIZE_MODEL](#)

SVM_REGRESSOR

Trains the SVM model on an input table or view. You can view the model using [SUMMARIZE_MODEL](#).

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
SVM_REGRESSOR ( 'model_name', 'input_relation', 'response_column', 'predictor_columns'  
                [USING PARAMETERS [exclude_columns='col1, col2, ... coln',]  
                [error_tolerance = value,]  
                [C=value,]  
                [epsilon= value,]  
                [max_iterations= value] ] )
```

Arguments

<i>model_name</i>	The name of the model. Model names are case-insensitive.
<i>input_relation</i>	The table or view that contains the training data.
<i>response_column</i>	The name of the column in <i>input_relation</i> that represents the dependent variable, or outcome. Valid Types: <ul style="list-style-type: none">• FLOAT• INT• NUMERIC
<i>predictor_columns</i>	A comma-separated list of the columns in <i>input_relation</i> that represent the independent variables for the model. Valid Types:

	<ul style="list-style-type: none"> • FLOAT • INT • NUMERIC <p>If the column name contains special characters, it must use double quotes.</p>
--	---

Parameters

<code>exclude_columns='col1, col2, ... coln'</code>	<p>(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>input_columns</code> argument.</p> <p>Default Value: Empty</p>
<code>error_tolerance=value</code>	<p>(Optional) Defines the acceptable error margin. Any data points outside this region add a penalty to the cost function.</p> <p>Default Value: 1.0</p>
<code>C=value</code>	<p>(Optional) Sets the weight for misclassification cost. The algorithm minimizes the regularization cost and the misclassification cost.</p> <p>Default Value: 1.0</p>
<code>epsilon=value</code>	<p>(Optional) Used to control accuracy.</p> <p>Default Value: 1e-3</p>
<code>max_iterations=value</code>	<p>(Optional) Determines the maximum number of iterations that the algorithm performs before achieving the specified accuracy result.</p> <p>Default Value: 100</p>

Privileges

To use `SVM_REGRESSOR`, you must either be a superuser or have `CREATE` privileges for the schema of the output view and `SELECT` privileges for the input table or view. There are no

privileges needed on the function itself.

See [GRANT \(Schema\)](#) and [GRANT \(Table\)](#).

Examples

This example shows how you can use the `SVM_REGRESSOR` function on the `faithful` table:

```
=> SELECT SVM_REGRESSOR('mySvmRegModel', 'faithful', 'eruptions', 'waiting'  
                        USING PARAMETERS error_tolerance=0.1, max_iterations=100);  
  
SVM_REGRESSOR  
-----  
Finished in 5 iterations.  
Accepted Rows: 272 Rejected Rows: 0  
(1 row)
```

See Also

- [Building an SVM for Regression Model](#)
- [SVM \(Support Vector Machine\) for Regression](#)
- [PREDICT_SVM_REGRESSOR](#)
- [SUMMARIZE_MODEL](#)

Mathematical Functions

Some of these functions are provided in multiple forms with different argument types. Except where noted, any given form of a function returns the same data type as its argument. The functions working with `DOUBLE PRECISION` data could vary in accuracy and behavior in boundary cases depending on the host system.

ABS

Returns the absolute value of the argument. The return value has the same data type as the argument..

Behavior Type

Immutable

Syntax

`ABS (expression)`

Parameters

<i>expression</i>	Is a value of type <code>INTEGER</code> or <code>DOUBLE PRECISION</code>
-------------------	--

Examples

```
SELECT ABS(-28.7);
 abs
-----
 28.7
(1 row)
```

ACOS

Returns a DOUBLE PRECISION value representing the trigonometric inverse cosine of the argument.

Behavior Type

Immutable

Syntax

`ACOS (expression)`

Parameters

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

Example

```
SELECT ACOS (1);
acos
-----
0
(1 row)
```

ASIN

Returns a DOUBLE PRECISION value representing the trigonometric inverse sine of the argument.

Behavior Type

Immutable

Syntax

`ASIN (expression)`

Parameters

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

Example

```
SELECT ASIN(1);
      asin
-----
1.5707963267949
(1 row)
```

ATAN

Returns a DOUBLE PRECISION value representing the trigonometric inverse tangent of the argument.

Behavior Type

Immutable

Syntax

`ATAN (expression)`

Parameters

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

Example

```
SELECT ATAN(1);
       atan
-----
0.785398163397448
(1 row)
```

ATAN2

Returns a DOUBLE PRECISION value representing the trigonometric inverse tangent of the arithmetic dividend of the arguments.

Behavior Type

Immutable

Syntax

```
ATAN2 ( quotient, divisor )
```

Parameters

<i>quotient</i>	Is an expression of type DOUBLE PRECISION representing the quotient
<i>divisor</i>	Is an expression of type DOUBLE PRECISION representing the divisor

Example

```
SELECT ATAN2(2,1);
       ATAN2
-----
1.10714871779409
(1 row)
```

CBRT

Returns the cube root of the argument. The return value has the type DOUBLE PRECISION.

Behavior Type

Immutable

Syntax

```
CBRT ( expression )
```

Parameters

<i>expression</i>	Value of type DOUBLE PRECISION
-------------------	--------------------------------

Examples

```
SELECT CBRT(27.0);  
  cbrt  
-----  
     3  
(1 row)
```

CEILING (CEIL)

Rounds the returned value up to the next whole number. Any expression that contains even a slight decimal is rounded up.

Behavior Type

Immutable

Syntax

```
CEILING ( expression ) CEIL ( expression )
```

Parameters

<i>expression</i>	Is a value of type INTEGER or DOUBLE PRECISION
-------------------	--

Notes

CEILING is the opposite of [FLOOR](#), which rounds the returned value down:

```
=> SELECT CEIL(48.01) AS ceiling, FLOOR(48.01) AS floor;
   ceiling | floor
-----+-----
         49 |    48
(1 row)
```

Examples

```
=> SELECT CEIL(-42.8);
   CEIL
-----
    -42
(1 row)

SELECT CEIL(48.01);
   CEIL
-----
     49
(1 row)
```

COS

Returns a DOUBLE PRECISION value that represents the trigonometric cosine of the passed parameter.

Behavior Type

Immutable

Syntax

`COS (expression)`

Parameters

<i>expression</i>	An expression of type DOUBLE PRECISION
-------------------	--

Example

```
SELECT COS(-1);
      COS
-----
0.54030230586814
(1 row)
```

COSH

Returns a DOUBLE PRECISION value that represents the hyperbolic cosine of the passed parameter.

Behavior Type

Immutable

Syntax

`COSH (expression)`

Parameters

<i>expression</i>	An expression of type DOUBLE PRECISION
-------------------	--

Example

```
=> SELECT COSH(-1);
      COSH
-----
1.54308063481524
```

COT

Returns a DOUBLE PRECISION value representing the trigonometric cotangent of the argument.

Behavior Type

Immutable

Syntax

COT (*expression*)

Parameters

<i>expression</i>	Is a value of type DOUBLE PRECISION
-------------------	-------------------------------------

Example

```
SELECT COT(1);
      cot
-----
0.642092615934331
(1 row)
```

DEGREES

Converts an expression from [RADIAN](#)S to fractional degrees, or from degrees, minutes, and seconds to fractional degrees. The return value has the type DOUBLE PRECISION.

Behavior Type

Immutable

Syntax 1

DEGREES (*radians*)

Syntax 2

DEGREES (*degrees, minutes, seconds*)

Parameters

<i>radians</i>	A unit of angular measure, 2π radians is equal to a full rotation.
<i>degrees</i>	A unit of angular measure, equal to 1/360 of a full rotation.
<i>minutes</i>	A unit of angular measurement, representing 1/60 of a degree.
<i>seconds</i>	A unit of angular measurement, representing 1/60 of a minute.

Examples

```
SELECT DEGREES(0.5);
      DEGREES
-----
28.6478897565412
(1 row)

SELECT DEGREES(1,2,3);
      DEGREES
-----
1.034166666666667
(1 row)
```

DISTANCE

Returns the distance (in kilometers) between two points. You specify the latitude and longitude of both the starting point and the ending point. You can also specify the radius of curvature for

greater accuracy when using an ellipsoidal model.

Behavior Type

Immutable

Syntax

```
DISTANCE ( Lat0, Lon0, Lat1, Lon1, radius_of_curvature )
```

Parameters

<i>Lat0</i>	Specifies the latitude of the starting point.
<i>Lon0</i>	Specifies the longitude of the starting point.
<i>Lat1</i>	Specifies the latitude of the ending point
<i>Lon1</i>	Specifies the longitude of the ending point.
<i>radius_of_curvature</i>	Specifies the radius of the curvature of the earth at the midpoint between the starting and ending points. This parameter allows for greater accuracy when using an ellipsoidal earth model. If you do not specify this parameter, it defaults to the WGS-84 average r1 radius, about 6371.009 km.

Example

This example finds the distance in kilometers for 1 degree of longitude at latitude 45 degrees, assuming earth is spherical.

```
SELECT DISTANCE(45,0, 45,1);
      DISTANCE
-----
78.6262959272162
(1 row)
```

DISTANCEV

Returns the distance (in kilometers) between two points using the Vincenty formula. Because the Vincenty formula includes the parameters of the WGS-84 ellipsoid model, you need not specify a radius of curvature. You specify the latitude and longitude of both the starting point and the ending point. This function is more accurate, but will be slower, than the `DISTANCE` function.

Behavior Type

Immutable

Syntax

```
DISTANCEV (lat0, lon0, lat1, lon1);
```

Parameters

<i>lat0</i>	Specifies the latitude of the starting point.
<i>lon0</i>	Specifies the longitude of the starting point.
<i>lat1</i>	Specifies the latitude of the ending point.
<i>lon1</i>	Specifies the longitude of the ending point.

Example

This example finds the distance in kilometers for 1 degree of longitude at latitude 45 degrees, assuming earth is ellipsoidal.

```
SELECT DISTANCEV(45,0, 45,1);
       distanceV
-----
78.8463347095916
(1 row)
```

EXP

Returns the exponential function, e to the power of a number. The return value has the same data type as the argument.

Behavior Type

Immutable

Syntax

EXP (*exponent*)

Parameters

<i>exponent</i>	Is an expression of type INTEGER or DOUBLE PRECISION
-----------------	--

Example

```
SELECT EXP(1.0);
      exp
-----
2.71828182845905
(1 row)
```

FLOOR

Rounds the returned value down to the next whole number. For example, each of these functions evaluates to 5:

```
floor(5.01)
floor(5.5)
floor(5.99)
```

Behavior Type

Immutable

Syntax

FLOOR (*expression*)

Parameters

<i>expression</i>	Is an expression of type INTEGER or DOUBLE PRECISION.
-------------------	---

Notes

FLOOR is the opposite of [CEILING](#), which rounds the returned value up:

```
=> SELECT FLOOR(48.01) AS floor, CEIL(48.01) AS ceiling;
 floor | ceiling
-----+-----
    48 |      49
(1 row)
```

Examples

```
=> SELECT FLOOR((TIMESTAMP '2005-01-17 10:00' - TIMESTAMP '2005-01-01') / INTERVAL '7');
 FLOOR
-----
      2
(1 row)

=> SELECT FLOOR(-42.8);
 FLOOR
-----
    -43
(1 row)

=> SELECT FLOOR(42.8);
 FLOOR
-----
     42
(1 row)
```

Although the following example looks like an INTEGER, the number on the left is 2^{49} as an INTEGER, but the number on the right is a FLOAT:

```
=> SELECT 1<<49, FLOOR(1 << 49);
   ?column?      |      floor
-----+-----
 562949953421312 | 562949953421312
(1 row)
```

Compare the above example to:

```
=> SELECT 1<<50, FLOOR(1 << 50);
   ?column?      |      floor
-----+-----
1125899906842624 | 1.12589990684262e+15
(1 row)
```

HASH

Calculates a hash value over the function arguments, producing a value in the range $0 \leq x < 2^{63}$.

The HASH function is typically used to segment a projection over a set of cluster nodes. The function selects a specific node for each row based on the values of the row columns. The HASH function distributes data evenly across the cluster, which facilitates optimal query execution.

Behavior Type

Immutable

Syntax

```
HASH ( expression-arg[,... ] )
```

Parameters

<i>expression-arg</i>	An expression of any data type, up to 32 arguments. Any functions that are included in <i>expression</i> must be deterministic. For the purpose of hash segmentation, each expression typically resolves to a column reference .
-----------------------	--

Examples

```
=> SELECT HASH(product_price, product_cost) FROM product_dimension
      WHERE product_price = '11';
      hash
-----
4157497907121511878
1799398249227328285
3250220637492749639
(3 rows)
```

See Also

[Hash Segmentation Clause](#)

LN

Returns the natural logarithm of the argument. The return data type is the same as the argument.

Behavior Type

Immutable

Syntax

LN (*expression*)

Parameters

<i>expression</i>	Is an expression of type INTEGER or DOUBLE PRECISION
-------------------	--

Example

```
SELECT LN(2);
ln
```

```
-----  
0.693147180559945  
(1 row)
```

LOG

Returns the logarithm to the specified base of the argument. The data type of the return value is the same data type as the passed parameter.

Behavior Type

Immutable

Syntax

```
LOG ( [ base, ] expression )
```

Parameters

<i>base</i>	Specifies the base (default is base 10)
<i>expression</i>	An expression of type INTEGER or DOUBLE PRECISION

Examples

```
=> SELECT LOG(2.0, 64);  
LOG  
-----  
6  
(1 row)  
  
SELECT LOG(100);  
LOG  
-----  
2  
(1 row)
```

LOG10

Returns the base 10 logarithm of the argument, also known as the *common logarithm*. The data type of the return value is the same as the data type of the passed parameter.

Behavior Type

Immutable

Syntax

```
LOG10 ( expression )
```

Parameters

<i>expression</i>	An expression of type INTEGER or DOUBLE PRECISION
-------------------	---

Examples

```
=> SELECT LOG10(30);
      LOG10
-----
 1.47712125471966
(1 row)
```

MOD

Returns the remainder of a division operation. MOD is also called `modulo`.

Behavior Type

Immutable

Syntax

`MOD(expression1, expression2)`

Parameters

<i>expression1</i>	Specifies the dividend (INTEGER, NUMERIC, or FLOAT)
<i>expression2</i>	Specifies the divisor (type same as dividend)

Notes

When computing `mod(N,M)`, the following rules apply:

- If either N or M is the null value, then the result is the null value.
- If M is zero, then an exception condition is raised: data exception — division by zero.
- Otherwise, the result is the unique exact numeric value R with scale 0 (zero) such that all of the following are true:
 - R has the same sign as N.
 - The absolute value of R is less than the absolute value of M.
 - $N = M * K + R$ for some exact numeric value K with scale 0 (zero).

Examples

```
SELECT MOD(9,4);
mod
-----
1
(1 row)

SELECT MOD(10,3);
mod
-----
1
(1 row)

SELECT MOD(-10,3);
mod
```

```
-----  
-1  
(1 row)  
SELECT MOD(-10,-3);  
mod  
-----  
-1  
(1 row)  
SELECT MOD(10,-3);  
mod  
-----  
1  
(1 row)
```

MOD(<float>, 0) gives an error:

```
=> SELECT MOD(6.2,0);  
ERROR: numeric division by zero
```

PI

Returns the constant pi (Π), the ratio of any circle's circumference to its diameter in Euclidean geometry. The return type is DOUBLE PRECISION.

Behavior Type

Immutable

Syntax

PI()

Examples

```
SELECT PI();  
pi  
-----  
3.14159265358979  
(1 row)
```

POWER (or POW)

Returns a DOUBLE PRECISION value representing one number raised to the power of another number. You can use either POWER or POW as the function name.

Behavior Type

Immutable

Syntax

```
POWER ( expression1, expression2 )
```

Parameters

<i>expression1</i>	Is an expression of type DOUBLE PRECISION that represents the base
<i>expression2</i>	Is an expression of type DOUBLE PRECISION that represents the exponent

Example

```
SELECT POWER(9.0, 3.0);
power
-----
    729
(1 row)
```

RADIANS

Returns a DOUBLE PRECISION value representing an angle expressed in radians. You can express the input angle in [DEGREES](#), and optionally include minutes and seconds.

Behavior Type

Immutable

Syntax

RADIANS (*degrees* [, *minutes*, *seconds*])

Parameters

<i>degrees</i>	A unit of angular measurement, representing 1/360 of a full rotation.
<i>minutes</i>	A unit of angular measurement, representing 1/60 of a degree.
<i>seconds</i>	A unit of angular measurement, representing 1/60 of a minute.

Examples

```
SELECT RADIANS(45);
       RADIANS
-----
0.785398163397448
(1 row)

SELECT RADIANS (1,2,3);
       RADIANS
-----
0.018049613347708
(1 row)
```

RANDOM

Returns a uniformly-distributed random number x , where $0 \leq x < 1$.

Typical pseudo-random generators accept a seed, which is set to generate a reproducible pseudo-random sequence. Vertica, however, distributes SQL processing over a cluster of nodes, where each node generates its own independent random sequence.

Results depending on RANDOM are not reproducible because the work might be divided differently across nodes. Therefore, Vertica automatically generates truly random seeds for each node each time a request is executed and does not provide a mechanism for forcing a specific seed.

Behavior Type

Volatile

Syntax

```
RANDOM()
```

Parameters

RANDOM has no arguments. Its result is a FLOAT8 data type (also called [DOUBLE PRECISION](#)).

Examples

In the following example, the result is a float, which is ≥ 0 and < 1.0 :

```
SELECT RANDOM();
       random
-----
0.211625560652465
(1 row)
```

RANDOMINT

Accepts and returns an INT8 value. `RANDOMINT(n)` returns one of the *n* integers from 0 through *n* - 1.

Typical pseudo-random generators accept a seed, which is set to generate a reproducible pseudo-random sequence. Vertica, however, distributes SQL processing over a cluster of nodes, where each node generates its own independent random sequence.

Results depending on RANDOM are not reproducible because the work might be divided differently across nodes. Therefore, Vertica automatically generates truly random seeds for each node each time a request is executed and does not provide a mechanism for forcing a specific seed.

Behavior Type

Volatile

Syntax

```
RANDOMINT ( n )
```

Parameters

The value accepted is any positive integer (*n*) between the values 1 and 9,223,372,036,854,775,807.

For general information on integer data types, refer to the section, [INTEGER](#).

Restrictions

If you provide a negative value, or if you exceed the maximum value, Vertica returns an error.

Example

In the following example, the result is an INT8, which is ≥ 0 and $< n$, randomly chosen from the set {0,1,2,3,4}.

```
=> SELECT RANDOMINT(5);
RANDOMINT
-----
          3
(1 row)
```

RANDOMINT_crypto

Accepts and returns an INT8 value. `RANDOMINT_crypto (n)` returns one of the *n* integers from 0 through *n* - 1. For this cryptographic random number generator, Vertica uses `RAND_` bytes to provide the random value.

Behavior Type

Volatile

Syntax

```
RANDOMINT_crypto ( n )
```

Parameters

The value accepted is any positive integer (n) between the values 1 and 9,223,372,036,854,775,807.

For general information on integer data types, see [INTEGER](#).

Restrictions

If you provide a negative value, or if you exceed the maximum value, Vertica returns an error.

Examples

In the following example, notice that the result is an INT8, which is ≥ 0 and $< n$, randomly chosen from the set {0,1,2,3,4}.

```
=> SELECT RANDOMINT_crypto(5);
RANDOMINT_crypto
-----
                3
(1 row)
```

ROUND

Rounds a value to a specified number of decimal places, retaining the original precision and scale. Fractions greater than or equal to .5 are rounded up. Fractions less than .5 are rounded down (truncated).

Behavior Type

Immutable

Syntax

```
ROUND ( expression [ , places ] )
```

Parameters

<i>expression</i>	Is an expression of type NUMERIC or DOUBLE PRECISION (FLOAT).
<i>places</i>	An INTEGER value. When places is a positive integer, Vertica rounds the value to the right of the decimal point. When places is a negative integer, Vertica rounds the value on the left side of the decimal point.

Notes

Using ROUND with a NUMERIC datatype returns NUMERIC, retaining the original precision and scale.

```
=> SELECT ROUND(3.5);  
ROUND  
-----  
    4.0  
(1 row)
```

Examples

```
=> SELECT ROUND(2.0, 1.0) FROM dual;  
ROUND  
-----  
    2.0  
(1 row)  
  
=> SELECT ROUND(12.345, 2.0);  
ROUND  
-----  
   12.350  
(1 row)  
  
=> SELECT ROUND(3.4444444444444444);  
ROUND  
-----  
  3.0000000000000000  
(1 row)  
  
=> SELECT ROUND(3.14159, 3);  
ROUND
```

```

-----
 3.14200
(1 row)

=> SELECT ROUND(1234567, -3);
      ROUND
-----
1235000
(1 row)

=> SELECT ROUND(3.4999, -1);
      ROUND
-----
 0.0000
(1 row)

```

The following example creates a table with two columns, adds one row of values, and shows sample rounding to the left and right of a decimal point.

```

=> CREATE TABLE sampleround (roundcol1 NUMERIC, roundcol2 NUMERIC);
CREATE TABLE

=> INSERT INTO sampleround VALUES (1234567, .1234567);
      OUTPUT
-----
          1
(1 row)

=> SELECT ROUND(roundcol1,-3) AS pn3, ROUND(roundcol1,-4) AS pn4, ROUND(roundcol1,-5) AS pn5 FROM
sampleround;

          pn3          |          pn4          |          pn5
-----+-----+-----
1235000.00000000000000 | 1230000.00000000000000 | 1200000.00000000000000
(1 row)

=> SELECT ROUND(roundcol2,3) AS p3, ROUND(roundcol2,4) AS p4, ROUND(roundcol2,5) AS p5 FROM
sampleround;

          p3          |          p4          |          p5
-----+-----+-----
 0.1230000000000000 | 0.1235000000000000 | 0.1234600000000000
(1 row)

```

SIGN

Returns a DOUBLE PRECISION value of -1, 0, or 1 representing the arithmetic sign of the argument.

Behavior Type

Immutable

Syntax

`SIGN (expression)`

Parameters

<i>expression</i>	Is an expression of type DOUBLE PRECISION
-------------------	---

Examples

```
SELECT SIGN(-8.4);
 sign
-----
   -1
(1 row)
```

SIN

Returns a DOUBLE PRECISION value that represents the trigonometric sine of the passed parameter.

Behavior Type

Immutable

Syntax

`SIN (expression)`

Parameters

<i>expression</i>	An expression of type DOUBLE PRECISION
-------------------	--

Example

```
SELECT SIN(30 * 2 * 3.14159 / 360);
      SIN
-----
0.4999999616987256
(1 row)
```

SINH

Returns a DOUBLE PRECISION value that represents the hyperbolic sine of the passed parameter.

Behavior Type

Immutable

Syntax

`SINH (expression)`

Parameters

<i>expression</i>	An expression of type DOUBLE PRECISION
-------------------	--

Example

```
=> SELECT SINH(30 * 2 * 3.14159 / 360);
      SINH
-----
0.547852969600632
```

SQRT

Returns a DOUBLE PRECISION value representing the arithmetic square root of the argument.

Behavior Type

Immutable

Syntax

`SQRT (expression)`

Parameters

<i>expression</i>	Is an expression of type DOUBLE PRECISION
-------------------	---

Examples

```
SELECT SQRT(2);
       sqrt
-----
1.4142135623731
(1 row)
```

TAN

Returns a DOUBLE PRECISION value that represents the trigonometric tangent of the passed parameter.

Behavior Type

Immutable

Syntax

`TAN (expression)`

Parameters

<i>expression</i>	An expression of type DOUBLE PRECISION
-------------------	--

Example

```
=> SELECT TAN(30);  
      TAN  
-----  
-6.40533119664628  
(1 row)
```

TANH

Returns a DOUBLE PRECISION value that represents the hyperbolic tangent of the passed parameter.

Behavior Type

Immutable

Syntax

```
TANH ( expression )
```

Parameters

<i>expression</i>	An expression of type DOUBLE PRECISION
-------------------	--

Example

```
=> SELECT TANH(-1);  
      TANH  
-----
```

```
-0.761594155955765
```

TRUNC

Returns the *expression* value fully truncated (toward zero). Supplying a *places* argument truncates the expression to the number of decimal places you indicate.

Behavior Type

Immutable

Syntax

```
TRUNC ( expression [ , places ] )
```

Parameters

<i>expression</i>	Is an expression of type NUMERIC or DOUBLE PRECISION (FLOAT).
<i>places</i>	An INTEGER value. When places is a positive integer, Vertica truncates the value to the right of the decimal point. When places is a negative integer, Vertica truncates the value on the left side of the decimal point.

Notes

Using TRUNC with a NUMERIC datatype returns NUMERIC, retaining the original precision and scale.

```
=> SELECT TRUNC(3.5);  
TRUNC  
-----  
3.0  
(1 row)
```

Examples

```
=> SELECT TRUNC(42.8);
TRUNC
-----
 42.0
(1 row)

=> SELECT TRUNC(42.4382, 2);
TRUNC
-----
42.4300
(1 row)
```

The following example creates a table with two columns, adds one row of values, and shows sample truncating to the left and right of a decimal point.

```
=> CREATE TABLE sampletrunc (truncol1 NUMERIC, truncol2 NUMERIC);
CREATE TABLE

=> INSERT INTO sampletrunc VALUES (1234567, .1234567);
OUTPUT
-----
      1
(1 row)

=> SELECT TRUNC(truncol1,-3) AS p3, TRUNC(truncol1,-4) AS p4, TRUNC(truncol1,-5) AS p5 FROM
sampletrunc;

      p3          |          p4          |          p5
-----+-----+-----
1234000.00000000000000 | 1230000.00000000000000 | 1200000.00000000000000
(1 row)

=> SELECT TRUNC(truncol2,3) AS p3, TRUNC(truncol2,4) AS p4, TRUNC(truncol2,5) AS p5 FROM sampletrunc;

      p3          |          p4          |          p5
-----+-----+-----
0.1230000000000000 | 0.1234000000000000 | 0.1234500000000000
(1 row)
```

WIDTH_BUCKET

Constructs equiwidth histograms, in which the histogram range is divided into intervals (buckets) of identical sizes. In addition, values below the low bucket return 0, and values above the high bucket return bucket_count +1. Returns an integer value.

Behavior Type

Immutable

Syntax

```
WIDTH_BUCKET ( expression, hist_min, hist_max, bucket_count )
```

Parameters

<i>expression</i>	The expression for which the histogram is created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If <i>expression</i> evaluates to null, then the <i>expression</i> returns null.
<i>hist_min</i>	An expression that resolves to the low boundary of bucket 1. Must also evaluate to numeric or datetime values and cannot evaluate to null.
<i>hist_max</i>	An expression that resolves to the high boundary of bucket <i>bucket_count</i> . Must also evaluate to a numeric or datetime value and cannot evaluate to null.
<i>bucket_count</i>	An expression that resolves to a constant, indicating the number of buckets. This expression always evaluates to a positive INTEGER.

Notes

- WIDTH_BUCKET divides a data set into buckets of equal width. For example, Age = 0–20, 20–40, 40–60, 60–80. This is known as an equiwidth histogram.
- When using WIDTH_BUCKET pay attention to the minimum and maximum boundary values. Each bucket contains values equal to or greater than the base value of that bucket, so that age ranges of 0–20, 20–40, and so on, are actually 0–19.99 and 20–39.999.
- WIDTH_BUCKET accepts the following data types: (FLOAT and/or INT), (TIMESTAMP and/or DATE and/or TIMESTAMPTZ), or (INTERVAL and/or TIME).

Examples

The following example returns five possible values and has three buckets: 0 [Up to 100), 1 [100–300), 2 [300–500), 3 [500–700), and 4 [700 and up):

```
SELECT product_description, product_cost, WIDTH_BUCKET(product_cost, 100, 700, 3);
```

The following example creates a nine-bucket histogram on the `annual_income` column for customers in Connecticut who are female doctors. The results return the bucket number to an “Income” column, divided into eleven buckets, including an underflow and an overflow. Note that if customers had an annual incomes greater than the maximum value, they would be assigned to an overflow bucket, 10:

```
SELECT customer_name, annual_income, WIDTH_BUCKET (annual_income, 100000, 1000000, 9) AS "Income"  
FROM public.customer_dimension WHERE customer_state='CT'  
AND title='Dr.' AND customer_gender='Female' AND household_id < '1000'  
ORDER BY "Income";
```

In the following result set, the reason there is a bucket 0 is because buckets are numbered from 1 to `bucket_count`. Anything less than the given value of `hist_min` goes in bucket 0, and anything greater than the given value of `hist_max` goes in the bucket `bucket_count+1`. In this example, bucket 9 is empty, and there is no overflow. The value 12,283 is less than 100,000, so it goes into the underflow bucket.

customer_name	annual_income	Income
Joanna A. Nguyen	12283	0
Amy I. Nguyen	109806	1
Juanita L. Taylor	219002	2
Carla E. Brown	240872	2
Kim U. Overstreet	284011	2
Tiffany N. Reyes	323213	3
Rebecca V. Martin	324493	3
Betty . Roy	476055	4
Midori B. Young	462587	4
Martha T. Brown	687810	6
Julie D. Miller	616509	6
Julie Y. Nielson	894910	8
Sarah B. Weaver	896260	8
Jessica C. Nielson	861066	8

(14 rows)

See Also

- [NTILE \[Analytic\]](#)

NULL-handling Functions

NULL-handling functions take arguments of any type, and their return type is based on their argument types.

COALESCE

Returns the value of the first non-null expression in the list. If all expressions evaluate to null, then COALESCE returns null.

COALESCE conforms to the ANSI SQL-92 standard.

Behavior Type

Immutable

Syntax

```
COALESCE ( expression[,...] );
```

Example

```
=> SELECT product_description, COALESCE(
       lowest_competitor_price, highest_competitor_price, average_competitor_price) AS price
   FROM product_dimension LIMIT 10;
 product_description | price
-----+-----
Brand #313 corn muffins | 565
Brand #591 hot dogs | 323
Brand #1247 american cheese | 263
Brand #1510 whole milk | 183
Brand #2549 vegetable soup | 491
Brand #4520 catfish | 113
Brand #4684 onions | 56
Brand #6078 salmon | 195
Brand #6924 bananas | 80
Brand #7912 green peppers | 180
(10 rows)
```

See Also

- [CASE Expressions](#)
- [ISNULL](#)

IFNULL

Returns the value of the first non-null expression in the list.

IFNULL is an alias of [NVL](#).

Behavior Type

Immutable

Syntax

```
IFNULL ( expression1 , expression2 );
```

Parameters

- If *expression1* is null, then IFNULL returns *expression2*.
- If *expression1* is not null, then IFNULL returns *expression1*.

Notes

- [COALESCE](#) is the more standard, more general function.
- IFNULL is equivalent to ISNULL.
- IFNULL is equivalent to COALESCE except that IFNULL is called with only two arguments.
- ISNULL(*a*, *b*) is different from *x* IS NULL.
- The arguments can have any data type supported by Vertica.

- Implementation is equivalent to the CASE expression. For example:

```
CASE WHEN expression1 IS NULL THEN expression2  
ELSE expression1 END;
```

- The following statement returns the value 140:

```
SELECT IFNULL(NULL, 140) FROM employee_dimension;
```

- The following statement returns the value 60:

```
SELECT IFNULL(60, 90) FROM employee_dimension;
```

Examples

```
=> SELECT IFNULL (SCORE, 0.0) FROM TESTING;  
IFNULL  
-----  
100.0  
87.0  
.0  
.0  
.0  
(5 rows)
```

See Also

- [CASE Expressions](#)
- [COALESCE](#)
- [NVL](#)
- [ISNULL](#)

ISNULL

Returns the value of the first non-null expression in the list.

ISNULL is an alias of [NVL](#).

Behavior Type

Immutable

Syntax

```
ISNULL ( expression1 , expression2 );
```

Parameters

- If *expression1* is null, then ISNULL returns *expression2*.
- If *expression1* is not null, then ISNULL returns *expression1*.

Notes

- [COALESCE](#) is the more standard, more general function.
- ISNULL is equivalent to COALESCE except that ISNULL is called with only two arguments.
- ISNULL(*a*, *b*) is different from *x* IS NULL.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression. For example:

```
CASE WHEN expression1 IS NULL THEN expression2  
ELSE expression1 END;
```

- The following statement returns the value 140:

```
SELECT ISNULL(NULL, 140) FROM employee_dimension;
```

- The following statement returns the value 60:

```
SELECT ISNULL(60, 90) FROM employee_dimension;
```

Examples

```
SELECT product_description, product_price,  
ISNULL(product_cost, 0.0) AS cost  
FROM product_dimension;
```

product_description	product_price	cost
Brand #59957 wheat bread	405	207
Brand #59052 blueberry muffins	211	140
Brand #59004 english muffins	399	240
Brand #53222 wheat bread	323	94
Brand #52951 croissants	367	121
Brand #50658 croissants	100	94
Brand #49398 white bread	318	25
Brand #46099 wheat bread	242	3
Brand #45283 wheat bread	111	105
Brand #43503 jelly donuts	259	19

(10 rows)

See Also

- [CASE Expressions](#)
- [COALESCE](#)
- [NVL](#)

NULLIF

Compares two expressions. If the expressions are not equal, the function returns the first expression (expression1). If the expressions are equal, the function returns null.

Behavior Type

Immutable

Syntax

```
NULLIF( expression1, expression2 )
```

Parameters

<i>expression1</i>	Is a value of any data type.
<i>expression2</i>	Must have the same data type as <i>expr1</i> or a type that can be implicitly cast to match <i>expression1</i> . The result has the same type as <i>expression1</i> .

Examples

The following series of statements illustrates one simple use of the NULLIF function.

Creates a single-column table `t` and insert some values:

```
CREATE TABLE t (x TIMESTAMPTZ);
INSERT INTO t VALUES('2009-09-04 09:14:00-04');
INSERT INTO t VALUES('2010-09-04 09:14:00-04');
```

Issue a select statement:

```
SELECT x, NULLIF(x, '2009-09-04 09:14:00 EDT') FROM t;
      x      |      nullif
-----+-----
2009-09-04 09:14:00-04 |
2010-09-04 09:14:00-04 | 2010-09-04 09:14:00-04
SELECT NULLIF(1, 2);
NULLIF
-----
      1
(1 row)
SELECT NULLIF(1, 1);
NULLIF
-----
(1 row)
SELECT NULLIF(20.45, 50.80);
NULLIF
-----
    20.45
(1 row)
```

NULLIFZERO

Evaluates to NULL if the value in the column is 0.

Syntax

`NULLIFZERO(expression)`

Parameters

<i>expression</i>	(INTEGER, DOUBLE PRECISION, INTERVAL, or NUMERIC) Is the string to evaluate for 0 values.
-------------------	---

Example

The TESTING table below shows the test scores for 5 students. Note that test scores are missing for S. Robinson and K. Johnson (NULL values appear in the Score column.)

```
=> SELECT * FROM TESTING;
  Name      | Score
-----+-----
 J. Doe     |    100
 R. Smith   |     87
 L. White   |     0
 S. Robinson |
 K. Johnson |
(5 rows)
```

The SELECT statement below specifies that Vertica should return any 0 values in the Score column as Null. In the results, you can see that Vertica returns L. White's 0 score as Null.

```
=> SELECT Name, NULLIFZERO(Score) FROM TESTING;
  Name      | NULLIFZERO
-----+-----
 J. Doe     |    100
 R. Smith   |     87
 L. White   |
 S. Robinson |
 K. Johnson |
(5 rows)
```

NVL

Returns the value of the first non-null expression in the list.

Behavior Type

Immutable

Syntax

```
NVL ( expression1 , expression2 );
```

Parameters

- If *expression1* is null, then NVL returns *expression2*.
- If *expression1* is not null, then NVL returns *expression1*.

Notes

- [COALESCE](#) is the more standard, more general function.
- NVL is equivalent to COALESCE except that NVL is called with only two arguments.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression:

```
CASE WHEN expression1 IS NULL THEN expression2  
ELSE expression1 END;
```

Examples

expression1 is not null, so NVL returns *expression1*:

```
SELECT NVL('fast', 'database');  
nvl  
-----  
fast  
(1 row)
```

expression1 is null, so NVL returns *expression2*:

```
SELECT NVL(null, 'database');
nvl
-----
database
(1 row)
```

expression2 is null, so NVL returns expression1:

```
SELECT NVL('fast', null);
nvl
-----
fast
(1 row)
```

In the following example, expression1 (title) contains nulls, so NVL returns expression2 and substitutes 'Withheld' for the unknown values:

```
SELECT customer_name, NVL(title, 'Withheld') as title
FROM customer_dimension
ORDER BY title;
customer_name | title
-----+-----
Alexander I. Lang | Dr.
Steve S. Harris | Dr.
Daniel R. King | Dr.
Luigi I. Sanchez | Dr.
Duncan U. Carcetti | Dr.
Meghan K. Li | Dr.
Laura B. Perkins | Dr.
Samantha V. Robinson | Dr.
Joseph P. Wilson | Mr.
Kevin R. Miller | Mr.
Lauren D. Nguyen | Mrs.
Emily E. Goldberg | Mrs.
Darlene K. Harris | Ms.
Meghan J. Farmer | Ms.
Bettercare | Withheld
Ameristar | Withheld
Initech | Withheld
(17 rows)
```

See Also

- [CASE Expressions](#)
- [COALESCE](#)
- [ISNULL](#)
- [NVL2](#)

NVL2

Takes three arguments. If the first argument is not NULL, it returns the second argument, otherwise it returns the third argument. The data types of the second and third arguments are implicitly cast to a common type if they don't agree, similar to [COALESCE](#).

Behavior Type

Immutable

Syntax

```
NVL2 ( expression1 , expression2 , expression3 );
```

Parameters

- If *expression1* is not null, then NVL2 returns *expression2*.
- If *expression1* is null, then NVL2 returns *expression3*.

Notes

Arguments two and three can have any data type supported by Vertica.

Implementation is equivalent to the CASE expression:

```
CASE WHEN expression1 IS NOT NULL THEN expression2 ELSE expression3 END;
```

Examples

In this example, *expression1* is not null, so NVL2 returns *expression2*:

```
SELECT NVL2('very', 'fast', 'database');
nv12
-----
fast
(1 row)
```

In this example, *expression1* is null, so NVL2 returns *expression3*:

```
SELECT NVL2(null, 'fast', 'database');
nvl2
-----
database
(1 row)
```

In the following example, expression1 (title) contains nulls, so NVL2 returns expression3 ('Withheld') and also substitutes the non-null values with the expression 'Known':

```
SELECT customer_name, NVL2(title, 'Known', 'Withheld')
as title
FROM customer_dimension
ORDER BY title;
customer_name | title
-----+-----
Alexander I. Lang | Known
Steve S. Harris | Known
Daniel R. King | Known
Luigi I. Sanchez | Known
Duncan U. Carcetti | Known
Meghan K. Li | Known
Laura B. Perkins | Known
Samantha V. Robinson | Known
Joseph P. Wilson | Known
Kevin R. Miller | Known
Lauren D. Nguyen | Known
Emily E. Goldberg | Known
Darlene K. Harris | Known
Meghan J. Farmer | Known
Bettercare | Withheld
Ameristar | Withheld
Initech | Withheld
(17 rows)
```

See Also

- [CASE Expressions](#)
- [COALESCE](#)
- [COALESCE](#)

ZEROIFNULL

Evaluates to 0 if the column is NULL.

Syntax

ZEROIFNULL(*expression*)

Parameters

<i>expression</i>	(INTEGER, DOUBLE PRECISION, INTERVAL, or NUMERIC) Is the string to evaluate for NULL values.
-------------------	--

Example

The TESTING table below shows the test scores for 5 students. Note that L. White's score is 0, and that scores are missing for S. Robinson and K. Johnson.

```
=> SELECT * FROM TESTING;
  Name  | Score
-----+-----
 J. Doe |   100
 R. Smith |    87
 L. White |     0
 S. Robinson |
 K. Johnson |
(5 rows)
```

The next SELECT statement specifies that Vertica should return any Null values in the Score column as 0s. In the results, you can see that Vertica returns a 0 score for S. Robinson and K. Johnson.

```
=> SELECT Name, ZEROIFNULL (Score) FROM TESTING;
  Name  | ZEROIFNULL
-----+-----
 J. Doe |       100
 R. Smith |        87
 L. White |         0
 S. Robinson |         0
 K. Johnson |         0
(5 rows)
```

Pattern Matching Functions

Used with the [MATCH Clause](#), the Vertica pattern matching functions return additional data about the patterns found/output. For example, you can use these functions to return values representing the name of the event or pattern that matched the input row, the sequential number of the match, or a partition-wide unique identifier for the instance of the pattern that matched.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- Landed on the company home page.
- Navigated to the product page.
- Ran a search.
- Clicked a link from the search results.
- Made a purchase.

For examples that use this clickstream model, see [Event Series Pattern Matching](#) in Analyzing Data.

See Also

- [MATCH Clause](#)
- [Event Series Pattern Matching](#)

EVENT_NAME

Returns a VARCHAR value representing the name of the event that matched the row.

Syntax

EVENT_NAME()

Notes

Pattern matching functions must be used in [MATCH Clause](#) syntax; for example, if you call EVENT_NAME() on its own, Vertica returns the following error message:

```
=> SELECT event_name();  
ERROR: query with pattern matching function event_name must include a MATCH clause
```

Example

Note: This example uses the schema defined in [Event Series Pattern Matching](#) in Analyzing Data. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on `website2.com` and identifies patterns where the user landed on `website2.com` from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs the values for EVENT_NAME(), which is the name of the event that matched the row.

```
SELECT uid,  
       sid,  
       ts,  
       refurl,  
       pageurl,  
       action,  
       event_name()  
FROM clickstream_log  
MATCH  
  (PARTITION BY uid, sid ORDER BY ts  
  DEFINE  
    Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',  
    Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',  
    Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'  
  PATTERN  
    P AS (Entry Onsite* Purchase)  
  ROWS MATCH FIRST EVENT);
```

uid	sid	ts	refurl	pageurl	action	event_name
1	100	12:00:00	website1.com	website2.com/home	V	Entry
1	100	12:01:00	website2.com/home	website2.com/floby	V	Onsite
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	Onsite
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	Purchase

```
2 | 100 | 12:10:00 | website1.com | website2.com/home | V | Entry
2 | 100 | 12:11:00 | website2.com/home | website2.com/forks | V | Onsite
2 | 100 | 12:13:00 | website2.com/forks | website2.com/buy | P | Purchase
(7 rows)
```

See Also

- [MATCH Clause](#)
- [MATCH_ID](#)
- [PATTERN_ID](#)
- [Event Series Pattern Matching](#)

MATCH_ID

Returns a successful pattern match as an INTEGER value. The returned value is the ordinal position of a match within a partition.

Syntax

```
MATCH_ID()
```

Notes

Pattern matching functions must be used in [MATCH Clause](#) syntax; for example, if you call `MATCH_ID()` on its own, Vertica returns the following error message:

```
=> SELECT match_id();
ERROR: query with pattern matching function match_id must include a MATCH clause
```

Example

Note: This example uses the schema defined in [Event Series Pattern Matching](#) in Analyzing Data. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on a site called `website2.com` and identifies patterns where the user reached `website2.com` from another Web site (Entry in

the MATCH clause) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for the MATCH_ID(), which represents a sequential number of the match.

```
SELECT uid,
       sid,
       ts,
       refurl,
       pageurl,
       action,
       match_id()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
DEFINE
  Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
PATTERN
  P AS (Entry Onsite* Purchase)
ROWS MATCH FIRST EVENT);
```

uid	sid	ts	refurl	pageurl	action	match_id
1	100	12:00:00	website1.com	website2.com/home	V	1
1	100	12:01:00	website2.com/home	website2.com/floby	V	2
1	100	12:02:00	website2.com/floby	website2.com/shamwow	V	3
1	100	12:03:00	website2.com/shamwow	website2.com/buy	P	4
2	100	12:10:00	website1.com	website2.com/home	V	1
2	100	12:11:00	website2.com/home	website2.com/forks	V	2
2	100	12:13:00	website2.com/forks	website2.com/buy	P	3

(7 rows)

See Also

- [MATCH Clause](#)
- [EVENT_NAME](#)
- [PATTERN_ID](#)
- [Event Series Pattern Matching](#)

PATTERN_ID

Returns an integer value that is a partition-wide unique identifier for the instance of the pattern that matched.

Syntax

PATTERN_ID()

Notes

Pattern matching functions must be used in [MATCH Clause](#) syntax; for example, if call PATTERN_ID() on its own, Vertica returns the following error message:

```
=> SELECT pattern_id();  
ERROR: query with pattern matching function pattern_id must include a MATCH clause
```

Example

Note: This example uses the schema defined in [Event Series Pattern Matching](#) in Analyzing Data. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for PATTERN_ID(), which represents the partition-wide identifier for the instance of the pattern that matched.

```
SELECT uid,  
       sid,  
       ts,  
       refurl,  
       pageurl,  
       action,  
       pattern_id()  
FROM clickstream_log  
MATCH  
  (PARTITION BY uid, sid ORDER BY ts  
  DEFINE  
    Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',  
    Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',  
    Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'  
  PATTERN  
    P AS (Entry Onsite* Purchase)  
  ROWS MATCH FIRST EVENT);
```

```
uid | sid | ts | refurl | pageurl | action | pattern_id  
----+-----+----+-----+-----+-----+-----  
1 | 100 | 12:00:00 | website1.com | website2.com/home | V | 1
```

```
1 | 100 | 12:01:00 | website2.com/home | website2.com/floby | V | 1
1 | 100 | 12:02:00 | website2.com/floby | website2.com/shamwow | V | 1
1 | 100 | 12:03:00 | website2.com/shamwow | website2.com/buy | P | 1
2 | 100 | 12:10:00 | website1.com | website2.com/home | V | 1
2 | 100 | 12:11:00 | website2.com/home | website2.com/forks | V | 1
2 | 100 | 12:13:00 | website2.com/forks | website2.com/buy | P | 1
(7 rows)
```

See Also

- [MATCH Clause](#)
- [EVENT_NAME](#)
- [MATCH_ID](#)
- [Event Series Pattern Matching](#)

Regular Expression Functions

A regular expression lets you perform pattern matching on strings of characters. The regular expression syntax allows you to precisely define the pattern used to match strings, giving you much greater control than wildcard matching used in the [LIKE](#) predicate. The Vertica regular expression functions let you perform tasks such as determining if a string value matches a pattern, extracting a portion of a string that matches a pattern, or counting the number of times a pattern occurs within a string.

Vertica uses the [Perl Compatible Regular Expression \(PCRE\)](#) library to evaluate regular expressions. As its name implies, PCRE's regular expression syntax is compatible with the syntax used by the Perl 5 programming language. You can read [PCRE's documentation](#) about its library. However, if you are unfamiliar with using regular expressions, the [Perl Regular Expressions Documentation](#) is a good introduction.

Note: The regular expression functions only operate on valid UTF-8 strings. If you try using a regular expression function on a string that is not valid UTF-8, the query fails with an error. To prevent an error from occurring, use the [ISUTF8](#) function as an initial clause to ensure the strings you pass to the regular expression functions are valid UTF-8 strings. Alternatively, or you can use the 'b' argument to treat the strings as binary octets, rather than UTF-8 encoded strings.

ISUTF8

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as [REGEXP_LIKE](#), which expect UTF-8 characters by default.

ISUTF8 checks for invalid UTF8 byte sequences, according to UTF-8 rules:

- invalid bytes
- an unexpected continuation byte
- a start byte not followed by enough continuation bytes
- an Overload Encoding

The presence of an invalid UTF8 byte sequence results in a return value of false.

Syntax

```
ISUTF8( string );
```

Parameters

<i>string</i>	The string to test for UTF-8 compliance.
---------------	--

Examples

```
=> SELECT ISUTF8(E'\xC2\xBF'); -- UTF-8 INVERTED QUESTION MARK ISUTF8
-----
t
(1 row)
=> SELECT ISUTF8(E'\xC2\xC0'); -- UNDEFINED UTF-8 CHARACTER
ISUTF8
-----
f
(1 row)
```

REGEXP_COUNT

Returns the number times a regular expression matches a string.

Syntax

```
REGEXP_COUNT( string, pattern [, position [, regex_modifier ] ] )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If <i>string</i> exists in a <code>__raw__</code> column of a flex or columnar table, cast <i>string</i> to a LONG VARCHAR before searching for <i>pattern</i> .
<i>pattern</i>	The regular expression to search for within <i>string</i> . The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.

<i>position</i>	<p>[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the <i>n</i>th character you specify.</p> <p>Default value: 1</p>												
<i>regexp_modifier</i>	<p>[Optional] One or more single-character flags that modify how the regular expression finds matches in <i>string</i>:</p> <table border="1" data-bbox="440 619 1408 1493"> <tr> <td data-bbox="440 619 511 688">b</td> <td data-bbox="511 619 1408 688">Treat strings as binary octets, rather than UTF-8 characters.</td> </tr> <tr> <td data-bbox="440 688 511 758">c</td> <td data-bbox="511 688 1408 758">Force the match to be case sensitive (the default).</td> </tr> <tr> <td data-bbox="440 758 511 827">i</td> <td data-bbox="511 758 1408 827">Force the match to be case insensitive.</td> </tr> <tr> <td data-bbox="440 827 511 1066">m</td> <td data-bbox="511 827 1408 1066">Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td> </tr> <tr> <td data-bbox="440 1066 511 1220">n</td> <td data-bbox="511 1066 1408 1220">Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td> </tr> <tr> <td data-bbox="440 1220 511 1493">x</td> <td data-bbox="511 1220 1408 1493">Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</td> </tr> </table>	b	Treat strings as binary octets, rather than UTF-8 characters.	c	Force the match to be case sensitive (the default).	i	Force the match to be case insensitive.	m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.	n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.	x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.
b	Treat strings as binary octets, rather than UTF-8 characters.												
c	Force the match to be case sensitive (the default).												
i	Force the match to be case insensitive.												
m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.												
n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.												
x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.												

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

Count the number of occurrences of the substring *an* in the string "a man, a plan, a canal, Panama."

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an');
REGEXP_COUNT
-----
                4
(1 row)
```

Find the number of occurrences of the substring *an* in the string "a man, a plan, a canal: Panama" starting with the fifth character.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an',5);
REGEXP_COUNT
-----
                3
(1 row)
```

Find the number of occurrences of a substring containing a lower-case character followed by *an*. In the first example, do not use a modifier. In the second example, use the *i* modifier to force the regular expression to ignore case.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an');
REGEXP_COUNT
-----
                3
(1 row)

=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an', 1, 'i');

REGEXP_COUNT
-----
                4
```

REGEXP_ILIKE

Returns true if the string contains a match for the regular expression. This function is similar to the [LIKE-predicate](#), except that it uses a case insensitive regular expression, rather than simple wildcard character matching.

Syntax

```
REGEXP_ILIKE( string, pattern )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .
<i>pattern</i>	A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

This example creates a table containing several strings to demonstrate regular expressions.

1. Create a table (`longvc`) with a single, long varchar column (`body`) and insert data with some distinct characters:

```
=> CREATE table longvc(body long varchar (1048576));
CREATE TABLE

=> insert into longvc values ('На берегу пустынных волн');
=> insert into longvc values ('Voin syödä lasia, se ei vahingoita minua');
=> insert into longvc values ('私はガラスを食べられます。それは私を傷つけません。');
=> insert into longvc values ('Je peux manger du verre, ça ne me fait pas mal. ');
=> insert into longvc values ('zésbaésbaa');

=> SELECT * FROM longvc;
          body
-----
На берегу пустынных волн
Voin syödä lasia, se ei vahingoita minua
私はガラスを食べられます。それは私を傷つけません。
Je peux manger du verre, ça ne me fait pas mal.
zésbaésbaa
(5 rows)
```

2. Pattern match table rows containing a specific character ('ç'), added as part of Step 1:

```
=> SELECT * FROM longvc where regexp_ilike(body, 'c');
      body
-----
Je peux manger du verre, ça ne me fait pas mal.
(1 row)
```

3. Select all rows that contain the substring 'a':

```
=> SELECT * FROM longvc where regexp_ilike(body, 'a');
      body
-----
Je peux manger du verre, ça ne me fait pas mal.
Voin syödä lasia, se ei vahingoita minua
zésbaésbaa
(3 rows)
```

REGEXP_INSTR

Returns the starting or ending position in a string where a regular expression matches. This function returns 0 if no match for the regular expression is found in the string.

Syntax

```
REGEXP_INSTR( string, pattern [, position [, occurrence ... [, return_position [, regexp_modifier ]
... [, captured_subexp ] ] ] )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .
<i>pattern</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.
<i>position</i>	[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the <i>n</i> th character you specify.

	Default value: 1												
<i>occurrence</i>	<p>[Optional] Controls which occurrence of a pattern match in the string to return. By default, the function returns the position of the first matching substring. Use this parameter to find the position of subsequent matching substrings. For example, setting this parameter to 3 returns the position of the third substring that matches the pattern.</p> <p>Default value: 1</p>												
<i>return_position</i>	<p>[Optional] Sets the position within the string to return. Using the default position (0), the function returns the string position of the first character of the substring that matches the pattern. If you set <i>return_position</i> to 1, the function returns the position of the first character after the end of the matching substring.</p> <p>Default value: 0</p>												
<i>regexp_modifier</i>	<p>[Optional] One or more single-character flags that modify how the regular expression finds matches in <i>string</i>:</p> <table border="1"> <tr> <td>b</td> <td>Treat strings as binary octets, rather than UTF-8 characters.</td> </tr> <tr> <td>c</td> <td>Force the match to be case sensitive (the default).</td> </tr> <tr> <td>i</td> <td>Force the match to be case insensitive.</td> </tr> <tr> <td>m</td> <td>Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td> </tr> <tr> <td>n</td> <td>Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td> </tr> <tr> <td>x</td> <td>Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</td> </tr> </table>	b	Treat strings as binary octets, rather than UTF-8 characters.	c	Force the match to be case sensitive (the default).	i	Force the match to be case insensitive.	m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.	n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.	x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.
b	Treat strings as binary octets, rather than UTF-8 characters.												
c	Force the match to be case sensitive (the default).												
i	Force the match to be case insensitive.												
m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.												
n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.												
x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.												

<i>captured_subexp</i>	<p>[Optional] The captured subexpression whose position to return. By default, the function returns the position of the first character in <i>string</i> that matches the regular expression. If you set this value from 1 – 9, the function returns the subexpression captured by the corresponding set of parentheses in the regular expression. For example, setting this value to 3 returns the substring captured by the third set of parentheses in the regular expression.</p> <p>Default value: 0</p> <p>Note: The subexpressions are numbered left to right, based on the appearance of opening parenthesis, so nested regular expressions . For example, in the regular expression <code>\s*(\w+\s+(\w+))</code>, subexpression 1 is the one that captures everything but any leading whitespaces.</p>
------------------------	--

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

Find the first occurrence of a sequence of letters starting with the letter e and ending with the letter y in the phrase "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y');
REGEXP_INSTR
-----
                1
(1 row)
```

Find the first sequence of letters starting with the letter e and ending with the letter y in the string "easy come, easy go" starting at the second character (2) ."

```
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y',2);
REGEXP_INSTR
-----
                12
(1 row)
```

Find the second sequence of letters starting with the letter e and ending with the letter y in the string "easy come, easy go" starting at the first character.

```
=> SELECT REGEXP_INSTR('easy come, easy go', 'e\w*y', 1, 2);
   REGEXP_INSTR
-----
                12
(1 row)
```

Find the position of the first character after the first whitespace in the string "easy come, easy go."

```
=> SELECT REGEXP_INSTR('easy come, easy go', '\s', 1, 1, 1);
   REGEXP_INSTR
-----
                6
(1 row)
```

Find the position of the start of the third word in a string by capturing each word as a subexpression, and returning the third subexpression's start position.

```
=> SELECT REGEXP_INSTR('one two three', '(\w+)\s+(\w+)\s+(\w+)', 1, 1, 0, '', 3);
   REGEXP_INSTR
-----
                9
(1 row)
```

REGEXP_LIKE

Returns true if the string matches the regular expression. This function is similar to the [LIKE-predicate](#), except that it uses regular expressions rather than simple wildcard character matching.

Syntax

```
REGEXP_LIKE( string, pattern [, modifiers ] )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .
<i>pattern</i>	A string containing the regular expression to match against the string. The

	<p>syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.</p>												
<i>modifiers</i>	<p>[Optional] One or more single-character flags that modify how the regular expression finds matches in <i>string</i>:</p> <table border="1"> <tr> <td>b</td> <td>Treat strings as binary octets, rather than UTF-8 characters.</td> </tr> <tr> <td>c</td> <td>Force the match to be case sensitive (the default).</td> </tr> <tr> <td>i</td> <td>Force the match to be case insensitive.</td> </tr> <tr> <td>m</td> <td>Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td> </tr> <tr> <td>n</td> <td>Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td> </tr> <tr> <td>x</td> <td>Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</td> </tr> </table>	b	Treat strings as binary octets, rather than UTF-8 characters.	c	Force the match to be case sensitive (the default).	i	Force the match to be case insensitive.	m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.	n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.	x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.
b	Treat strings as binary octets, rather than UTF-8 characters.												
c	Force the match to be case sensitive (the default).												
i	Force the match to be case insensitive.												
m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.												
n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.												
x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.												

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

This example creates a table containing several strings to demonstrate regular expressions.

```
=> CREATE TABLE t (v VARCHAR);
CREATE TABLE
=> CREATE PROJECTION t1 AS SELECT * FROM t;
CREATE PROJECTION
=> COPY t FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> aaa
>> Aaa
>> abc
>> abc1
>> 123
>> \.
=> SELECT * FROM t;
   v
-----
aaa
Aaa
abc
abc1
123
(5 rows)
```

Select all records in the table that contain the letter "a."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a');
   v
-----
Aaa
aaa
abc
abc1
(4 rows)
```

Select all of the rows in the table that start with the letter "a."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '^a');
   v
-----
aaa
abc
abc1
(3 rows)
```

Select all rows that contain the substring "aa."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aa');
   v
-----
Aaa
aaa
(2 rows)
```

Select all rows that contain a digit.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '\d');
v
-----
123
abc1
(2 rows)
```

Select all rows that contain the substring "aaa."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa');
v
-----
aaa
(1 row)
```

Select all rows that contain the substring "aaa" using case insensitive matching.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa', 'i');
v
-----
Aaa
aaa
(2 rows)
```

Select rows that contain the substring "a b c."

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c');
v
---
(0 rows)
```

Select rows that contain the substring "a b c" ignoring space within the regular expression.

```
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c', 'x');
v
-----
abc
abc1
(2 rows)
```

Add multi-line rows to demonstrate using the "m" modifier.

```
=> COPY t FROM stdin RECORD TERMINATOR '!';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Record 1 line 1
>> Record 1 line 2
>> Record 1 line 3!
>> Record 2 line 1
>> Record 2 line 2
>> Record 2 line 3!
>> \.
```

Select rows that start with the substring "Record" and end with the substring "line 2."

```
=> SELECT v from t WHERE REGEXP_LIKE(v, '^Record.*line 2$');
v
---
(0 rows)
```

Select rows that start with the substring "Record" and end with the substring "line 2," treating multiple lines as separate strings.

```
=> SELECT v from t WHERE REGEXP_LIKE(v, '^Record.*line 2$', 'm');
v
-----
Record 2 line 1
Record 2 line 2
Record 2 line 3
Record 1 line 1
Record 1 line 2
Record 1 line 3
(2 rows)
```

REGEXP_NOT_ILIKE

Returns true if the string does not match the case-insensitive regular expression.

Syntax

```
REGEXP_NOT_ILIKE( string, pattern )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .
<i>pattern</i>	A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

This example creates a table containing strings to demonstrate regular expressions.

1. Create a table (`longvc`) with a single, long varchar column (`body`). Then, insert data with some distinct characters, and query the table contents:

```
=> CREATE table longvc(body long varchar (1048576));
CREATE TABLE

=> insert into longvc values ('На берегу пустынных волн');
=> insert into longvc values ('Voin syödä lasia, se ei vahingoita minua');
=> insert into longvc values ('私はガラスを食べられます。それは私を傷つけません。');
=> insert into longvc values ('Je peux manger du verre, ça ne me fait pas mal. ');
=> insert into longvc values ('zésbaésbaa');

=> SELECT * FROM longvc;
           body
-----
На берегу пустынных волн
Voin syödä lasia, se ei vahingoita minua
私はガラスを食べられます。それは私を傷つけません。
Je peux manger du verre, ça ne me fait pas mal.
zésbaésbaa
(5 rows)
```

2. Use `REGEXP_NOT_ILIKE` to return rows that do not contain a specific character ('ç'):

```
=> SELECT * FROM longvc where regexp_not_ilike(body, 'ç');
           body
-----
Voin syödä lasia, se ei vahingoita minua
zésbaésbaa
На берегу пустынных волн
私はガラスを食べられます。それは私を傷つけません。
(4 rows)
```

3. Pattern match all rows that do not contain the substring 'a':

```
=> SELECT * FROM longvc where regexp_not_ilike(body, 'a');
           body
-----
На берегу пустынных волн
私はガラスを食べられます。それは私を傷つけません。
(2 rows)
```

REGEXP_NOT_LIKE

Returns true if the string does not contain a match for the regular expression. This function is a case sensitive regular expression.

Syntax

```
REGEXP_NOT_LIKE( string, pattern modifiers ] )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .												
<i>pattern</i>	A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.												
<i>modifiers</i>	[Optional] One or more single-character flags that modify how the regular expression finds matches in <i>string</i> : <table border="1"><tr><td>b</td><td>Treat strings as binary octets, rather than UTF-8 characters.</td></tr><tr><td>c</td><td>Force the match to be case sensitive (the default).</td></tr><tr><td>i</td><td>Force the match to be case insensitive.</td></tr><tr><td>m</td><td>Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td></tr><tr><td>n</td><td>Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td></tr><tr><td>x</td><td>Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in</td></tr></table>	b	Treat strings as binary octets, rather than UTF-8 characters.	c	Force the match to be case sensitive (the default).	i	Force the match to be case insensitive.	m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.	n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.	x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in
b	Treat strings as binary octets, rather than UTF-8 characters.												
c	Force the match to be case sensitive (the default).												
i	Force the match to be case insensitive.												
m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.												
n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.												
x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in												

the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

These examples demonstrate the REGEXP_NOT_LIKE regular expression function.

1. Create a table (longvc) with a single, long varchar column (body). Then, insert data with some distinct characters, and query the table contents:

```
=> CREATE table longvc(body long varchar (1048576));
CREATE TABLE

=> insert into longvc values ('На берегу пустынных волн');
=> insert into longvc values ('Voin syödä lasia, se ei vahingoita minua');
=> insert into longvc values ('私はガラスを食べられます。それは私を傷つけません。');
=> insert into longvc values ('Je peux manger du verre, ça ne me fait pas mal. ');
=> insert into longvc values ('zésbaésbaa');

=> SELECT * FROM longvc;
          body
-----
На берегу пустынных волн
Voin syödä lasia, se ei vahingoita minua
私はガラスを食べられます。それは私を傷つけません。
Je peux manger du verre, ça ne me fait pas mal.
zésbaésbaa
(5 rows)
```

2. Use REGEXP_NOT_LIKE to return rows that do not contain a specific character ('ç'):

```
=> SELECT * FROM longvc where regexp_not_like(body, 'ç');
          body
-----
Voin syödä lasia, se ei vahingoita minua
zésbaésbaa
```

```
На берегу пустынных волн  
私はガラスを食べられます。それは私を傷つけません。  
(4 rows)
```

3. Return all rows that do not contain these characters ('.*ö.*ä'):

```
=> SELECT * FROM longvc where regexp_not_like(body, '.*ö.*ä');  
body  
-----  
Je peux manger du verre, ça ne me fait pas mal.  
zésbaésbaa  
На берегу пустынных волн  
私はガラスを食べられます。それは私を傷つけません。  
(4 rows)
```

4. Pattern match all rows that do not contain these specific characters ('z.*eśbaa'):

```
=> SELECT * FROM longvc where regexp_not_like(body, 'z.*eśbaa');  
body  
-----  
Je peux manger du verre, ça ne me fait pas mal.  
Voin syödä lasia, se ei vahingoita minua  
zésbaésbaa  
На берегу пустынных волн  
私はガラスを食べられます。それは私を傷つけません。  
(5 rows)
```

REGEXP_REPLACE

Replace all occurrences of a substring that match a regular expression with another substring. It is similar to the [REPLACE](#) function, except it uses a regular expression to select the substring to be replaced.

Syntax

```
REGEXP_REPLACE( string, target [, replacement [, position [, occurrence ... [, regexp_modifiers ] ] ] )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .
<i>target</i>	The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.

<i>replacement</i>	<p>The string to replace matched substrings. If you do not supply a <i>replacement</i>, the function deletes matched substrings. The replacement string can contain backreferences for substrings captured by the regular expression. The first captured substring is inserted into the replacement string using \1, the second \2, and so on.</p>												
<i>position</i>	<p>[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the <i>n</i>th character you specify.</p> <p>Default value: 1</p>												
<i>occurrence</i>	<p>[Optional] Controls which occurrence of a pattern match in the string to return. By default, the function returns the position of the first matching substring. Use this parameter to find the position of subsequent matching substrings. For example, setting this parameter to 3 returns the position of the third substring that matches the pattern.</p> <p>Default value: 1</p>												
<i>regexp_modifier</i>	<p>[Optional] One or more single-character flags that modify how the regular expression finds matches in <i>string</i>:</p> <table border="1" data-bbox="440 1150 1409 1864"> <tr> <td data-bbox="440 1150 509 1220">b</td> <td data-bbox="509 1150 1409 1220">Treat strings as binary octets, rather than UTF-8 characters.</td> </tr> <tr> <td data-bbox="440 1220 509 1289">c</td> <td data-bbox="509 1220 1409 1289">Force the match to be case sensitive (the default).</td> </tr> <tr> <td data-bbox="440 1289 509 1358">i</td> <td data-bbox="509 1289 1409 1358">Force the match to be case insensitive.</td> </tr> <tr> <td data-bbox="440 1358 509 1598">m</td> <td data-bbox="509 1358 1409 1598">Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td> </tr> <tr> <td data-bbox="440 1598 509 1751">n</td> <td data-bbox="509 1598 1409 1751">Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td> </tr> <tr> <td data-bbox="440 1751 509 1864">x</td> <td data-bbox="509 1751 1409 1864">Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and</td> </tr> </table>	b	Treat strings as binary octets, rather than UTF-8 characters.	c	Force the match to be case sensitive (the default).	i	Force the match to be case insensitive.	m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.	n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.	x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and
b	Treat strings as binary octets, rather than UTF-8 characters.												
c	Force the match to be case sensitive (the default).												
i	Force the match to be case insensitive.												
m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.												
n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.												
x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and												

comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Another key difference between Oracle and Vertica is that Vertica can handle an unlimited number of captured subexpressions, while Oracle is limited to nine.

In Vertica, you can use `\10` in the replacement pattern to access the substring captured by the tenth set of parentheses in the regular expression. In Oracle, `\10` is treated as the substring captured by the first set of parentheses, followed by a zero. To force this Oracle behavior in Vertica, use the `\g` back reference and enclose the number of the captured subexpression in curly braces. For example, `\g{1}0` is the substring captured by the first set of parentheses followed by a zero.

You can also name captured subexpressions to make your regular expressions less ambiguous. See the [PCRE](#) documentation for details.

Examples

Find groups of "word characters" (letters, numbers and underscore) ending with "thy" in the string "healthy, wealthy, and wise" and replace them with nothing.

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy');
   REGEXP_REPLACE
-----
, , and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy', 'something');
   REGEXP_REPLACE
-----
something, something, and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something" starting at the third character in the string.

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy', 'something', 3);
       REGEXP_REPLACE
-----
hesomething, something, and wise
(1 row)
```

Replace the second group of word characters ending with "thy" with the string "something."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy', 'something', 1, 2);
       REGEXP_REPLACE
-----
healthy, something, and wise
(1 row)
```

Find groups of word characters ending with "thy" capturing the letters before the "thy", and replace with the captured letters plus the letters "ish."

```
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','(\w+)thy', '\1ish');
       REGEXP_REPLACE
-----
healish, wealish, and wise
(1 row)
```

Create a table to demonstrate replacing strings in a query.

```
=> CREATE TABLE customers (name varchar(50), phone varchar(11));
CREATE TABLE
=> CREATE PROJECTION customers1 AS SELECT * FROM customers;
CREATE PROJECTION
=> COPY customers FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Able,Adam|17815551234
>> Baker,Bob|18005551111
>> Chu,Cindy|16175559876
>> Dodd,Dinara|15083452121
>> \.
```

Query the customers, using REGEXP_REPLACE to format the phone numbers.

```
=> SELECT name, REGEXP_REPLACE(phone, '(\d)(\d{3})(\d{3})(\d{4})',
'\1-(\2) \3-\4') as phone FROM customers;
   name      |      phone
-----+-----
Able, Adam   | 1-(781) 555-1234
Baker,Bob    | 1-(800) 555-1111
Chu,Cindy    | 1-(617) 555-9876
Dodd,Dinara  | 1-(508) 345-2121
(4 rows)
```

REGEXP_SUBSTR

Returns the substring that matches a regular expression within a string. If no matches are found, this function returns NULL. This is different from an empty string, which the function can return if the regular expression matches a zero-length string.

Syntax

```
REGEXP_SUBSTR( string, pattern [, position [, occurrence [, regexp_modifier... [, captured_subexp ] ] ] )
```

Parameters

<i>string</i>	The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <i>pattern</i> .
<i>pattern</i>	The regular expression to find a substring to extract. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.
<i>position</i>	[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the <i>n</i> th character you specify. Default value: 1
<i>occurrence</i>	[Optional] Controls which occurrence of a pattern match in the string to return. By default, the function returns the position of the first matching substring. Use this parameter to find the position of subsequent matching substrings. For example, setting this parameter to 3 returns the position of the third substring that matches the pattern. Default value: 1
<i>regexp_modifier</i>	[Optional] One or more single-character flags that modify how the regular expression finds matches in <i>string</i> :

	<table border="1"> <tr> <td data-bbox="440 212 509 279">b</td> <td data-bbox="509 212 1408 279">Treat strings as binary octets, rather than UTF-8 characters.</td> </tr> <tr> <td data-bbox="440 279 509 346">c</td> <td data-bbox="509 279 1408 346">Force the match to be case sensitive (the default).</td> </tr> <tr> <td data-bbox="440 346 509 413">i</td> <td data-bbox="509 346 1408 413">Force the match to be case insensitive.</td> </tr> <tr> <td data-bbox="440 413 509 653">m</td> <td data-bbox="509 413 1408 653">Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td> </tr> <tr> <td data-bbox="440 653 509 808">n</td> <td data-bbox="509 653 1408 808">Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td> </tr> <tr> <td data-bbox="440 808 509 1085">x</td> <td data-bbox="509 808 1408 1085">Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.</td> </tr> </table>	b	Treat strings as binary octets, rather than UTF-8 characters.	c	Force the match to be case sensitive (the default).	i	Force the match to be case insensitive.	m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.	n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.	x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.
b	Treat strings as binary octets, rather than UTF-8 characters.												
c	Force the match to be case sensitive (the default).												
i	Force the match to be case insensitive.												
m	Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line (\$) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.												
n	Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.												
x	Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.												
<p><i>captured_subexp</i></p>	<p>[Optional] The captured subexpression whose position to return. By default, the function returns the position of the first character in <i>string</i> that matches the regular expression. If you set this value from 1 – 9, the function returns the subexpression captured by the corresponding set of parentheses in the regular expression. For example, setting this value to 3 returns the substring captured by the third set of parentheses in the regular expression.</p> <p>Default value: 0</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Note: The subexpressions are numbered left to right, based on the appearance of opening parenthesis, so nested regular expressions. For example, in the regular expression <code>\s*(\w+\s+(\w+))</code>, subexpression 1 is the one that captures everything but any leading whitespaces.</p> </div>												

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

Select the first substring of letters that end with "thy."

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise', '\w+thy');
REGEXP_SUBSTR
-----
healthy
(1 row)
```

Select the first substring of letters that ends with "thy" starting at the second character in the string.

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise', '\w+thy', 2);
REGEXP_SUBSTR
-----
ealthy
(1 row)
```

Select the second substring of letters that ends with "thy."

```
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise', '\w+thy', 1, 2);
REGEXP_SUBSTR
-----
wealthy
(1 row)
```

Return the contents of the third captured subexpression, which captures the third word in the string.

```
=> SELECT REGEXP_SUBSTR('one two three', '(\w+)\s+(\w+)\s+(\w+)', 1, 1, ' ', 3);
REGEXP_SUBSTR
-----
three
(1 row)
```

Sequence Functions

The sequence functions provide simple, multiuser-safe methods for obtaining successive sequence values from sequence objects.

NEXTVAL

Returns the next value in a sequence. NEXTVAL is used in INSERT, COPY, and SELECT statements to create unique column values.

Call NEXTVAL after creating a sequence to initialize the sequence with its default value. Thereafter, call NEXTVAL to increment the sequence value for ascending sequences, or decrement its value for descending sequences.

Behavior Type

Volatile

Syntax

```
NEXTVAL(' [schema . ]sequence-name' )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>sequence-name</i>	Identifies the target sequence.

Privileges

- SELECT privilege on sequence
- USAGE privilege on sequence schema

Examples

The following example creates an ascending sequence called `my_seq`, starting at 101:

```
=> CREATE SEQUENCE my_seq START 101;
```

The following command generates the first number in the sequence:

```
=> SELECT NEXTVAL('my_seq');
nextval
-----
      101
(1 row)
```

The following command generates the next number in the sequence:

```
=> SELECT NEXTVAL('my_seq');
nextval
-----
      102
(1 row)
```

The following command illustrates how `NEXTVAL` is evaluated on a per-row basis, so in this example, both calls to `NEXTVAL` yield the same result:

```
=> SELECT NEXTVAL('my_seq'), NEXTVAL('my_seq');
nextval | nextval
-----+-----
      103 |      103
(1 row)
```

The following example illustrates how the `NEXTVAL` is always evaluated first (and here, increments the `my_seq` sequence from its previous value), even when `CURRVAL` precedes `NEXTVAL`:

```
=> SELECT CURRVAL('my_seq'), NEXTVAL('my_seq');
currval | nextval
-----+-----
      104 |      104
(1 row)
```

The following example shows how to use NEXTVAL in a table SELECT statement. Notice that the nextval column is incremented by 1 again:

```
=> SELECT NEXTVAL('my_seq'), product_description FROM product_dimension LIMIT 10;
nextval |      product_description
-----+-----
    105 | Brand #2 bagels
    106 | Brand #1 butter
    107 | Brand #6 chicken noodle soup
    108 | Brand #5 golf clubs
    109 | Brand #4 brandy
    110 | Brand #3 lamb
    111 | Brand #11 vanilla ice cream
    112 | Brand #10 ground beef
    113 | Brand #9 camera case
    114 | Brand #8 halibut
(10 rows)
```

See Also

- [ALTER SEQUENCE](#)
- [CREATE SEQUENCE](#)
- [CURRVAL](#)
- [DROP SEQUENCE](#)
- [Working with Sequence Types](#)
- [Sequence Privileges](#)

CURRVAL

Returns the last value across all nodes that was set by [NEXTVAL](#) on this sequence in the current session. If NEXTVAL was never called on this sequence since its creation, Vertica returns an error.

Behavior Type

Volatile

Syntax

```
CURRVAL(['schema.'sequence-name')
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>sequence-name</i>	The target sequence

Privileges

- SELECT privilege on sequence
- USAGE privilege on sequence schema

Examples

The following example creates an ascending sequence called sequential, starting at 101:

```
=> CREATE SEQUENCE seq2 START 101;
```

You can call CURRVAL only after you initiate the sequence with NEXTVAL; otherwise, Vertica returns an error:

```
=> SELECT CURRVAL('seq2');  
ERROR: Sequence seq2 has not been accessed in the session
```

Use NEXTVAL to generate the first number for this sequence:

```
=> SELECT NEXTVAL('seq2');  
nextval  
-----  
101  
(1 row)
```

Now you can use CURRVAL to return the current number from this sequence:

```
=> SELECT CURRVAL('seq2');
currval
-----
      101
(1 row)
```

The following command shows how to use CURRVAL in a SELECT statement:

```
=> CREATE TABLE customer3 (
  lname VARCHAR(25),
  fname VARCHAR(25),
  membership_card INTEGER,
  ID INTEGER
);
INSERT INTO customer3 VALUES ('Brown' , 'Sabra' , 072753, CURRVAL('my_seq'));
SELECT CURRVAL('seq2'), lname FROM customer3;
CURRVAL | lname
-----+-----
      101 | Brown
(1 row)
```

The following example illustrates how the NEXTVAL is always evaluated first (and here, increments the my_seq sequence from its previous value), even when CURRVAL precedes NEXTVAL:

```
=> SELECT CURRVAL('my_seq'), NEXTVAL('my_seq');
currval | nextval
-----+-----
      102 |      102
(1 row)
```

LAST_INSERT_ID

Returns the last value of a column whose value is automatically incremented through AUTO_INCREMENT or IDENTITY [column constraints](#). If multiple sessions concurrently load the same table, the returned value is the last value generated for an AUTO_INCREMENT column by an insert in that session.

Behavior Type

Volatile

Syntax

```
LAST_INSERT_ID()
```

Privileges

- Table owner
- USAGE privileges on schema

Restrictions

- This function works only with AUTO_INCREMENT and IDENTITY columns. See [column constraints](#) for [CREATE TABLE](#).
- LAST_INSERT_ID does not work with sequence generators created through [CREATE SEQUENCE](#).

Examples

1. Create table customer4:

```
=> CREATE TABLE customer4(  
    ID IDENTITY(2,2),  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER  
);  
=> INSERT INTO customer4(lname, fname, membership_card) VALUES ('Gupta', 'Saleem', 475987);
```

The IDENTITY column has a seed of 2, which specifies the value for the first row loaded into the table, and an increment of 2, which specifies the value that is added to the IDENTITY value of the previous row.

2. Query the table you just created:

```
=> SELECT * FROM customer4;  
ID | lname | fname | membership_card  
-----+-----+-----+-----  
2 | Gupta | Saleem | 475987  
(1 row)
```

3. Insert additional values:

```
=> INSERT INTO customer4(lname, fname, membership_card) VALUES ('Lee', 'Chen', 598742);
```

4. Call LAST_INSERT_ID:

```
=> SELECT LAST_INSERT_ID();
LAST_INSERT_ID
-----
                4
(1 row)
```

5. Query the table again:

```
=> SELECT * FROM customer4;
ID | lname | fname | membership_card
-----+-----+-----+-----
 2 | Gupta | Saleem |          475987
 4 | Lee   | Chen   |          598742
(2 rows)
```

6. Add another row:

```
=> INSERT INTO customer4(lname, fname, membership_card) VALUES ('Davis', 'Bill', 469543);
```

7. Call LAST_INSERT_ID:

```
=> SELECT LAST_INSERT_ID();
LAST_INSERT_ID
-----
                6
(1 row)
```

8. Query the table again:

```
=> SELECT * FROM customer4;
ID | lname | fname | membership_card
-----+-----+-----+-----
 2 | Gupta | Saleem |          475987
 4 | Lee   | Chen   |          598742
 6 | Davis | Bill   |          469543
(3 rows)
```

See Also

- [Working with Sequence Types](#)
- [Sequence Privileges](#)

String Functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

This section describes functions and operators for examining and manipulating string values. Strings in this context include values of the types CHAR, VARCHAR, BINARY, and VARBINARY.

Unless otherwise noted, all of the functions listed in this section work on all four data types. As opposed to some other SQL implementations, Vertica keeps CHAR strings unpadded internally, padding them only on final output. So converting a CHAR(3) 'ab' to VARCHAR(5) results in a VARCHAR of length 2, not one with length 3 including a trailing space.

Some of the functions described here also work on data of non-string types by converting that data to a string representation first. Some functions work only on character strings, while others work only on binary strings. Many work for both. BINARY and VARBINARY functions ignore multibyte UTF-8 character boundaries.

Non-binary character string functions handle normalized multibyte UTF-8 characters, as specified by the Unicode Consortium. Unless otherwise specified, those character string functions for which it matters can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of UTF-8 characters. This is accomplished by adding "USING OCTETS" or "USING CHARACTERS" (default) as a parameter to the function.

Some character string functions are stable because in general UTF-8 case-conversion, searching and sorting can be locale dependent. Thus, LOWER is stable, while LOWERB is immutable. The USING OCTETS clause converts these functions into their "B" forms, so they become immutable. If the locale is set to collation=binary, which is the default, all string functions—except CHAR_LENGTH/CHARACTER_LENGTH, LENGTH, SUBSTR, and OVERLAY—are converted to their "B" forms and so are immutable.

BINARY implicitly converts to VARBINARY, so functions that take VARBINARY arguments work with BINARY.

ASCII

Converts the first character of a VARCHAR datatype to an INTEGER.

Behavior Type

Immutable

Syntax

ASCII (*expression*)

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Notes

- ASCII is the opposite of the [CHR](#) function.
- ASCII operates on UTF-8 characters, not only on single-byte ASCII characters. It continues to get the same results for the ASCII subset of UTF-8.

Examples

This example returns employees whose last name begins with M. The ASCII equivalent of M is 77:

```
=> SELECT employee_last_name FROM employee_dimension
      WHERE ASCII(SUBSTR(employee_last_name, 1, 1)) = 76
      LIMIT 5;
employee_last_name
-----
Lewis
Lewis
Lampert
Lampert
Li
(5 rows)
```

BIT_LENGTH

Returns the length of the string expression in bits (bytes * 8) as an INTEGER.

Behavior Type

Immutable

Syntax

```
BIT_LENGTH ( expression )
```

Parameters

<i>expression</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string to convert.
-------------------	--

Notes

BIT_LENGTH applies to the contents of VARCHAR and VARBINARY fields.

Examples

Expression	Result
SELECT BIT_LENGTH('abc'::varbinary);	24
SELECT BIT_LENGTH('abc'::binary);	8
SELECT BIT_LENGTH(' '::varbinary);	0
SELECT BIT_LENGTH(' '::binary);	8
SELECT BIT_LENGTH(null::varbinary);	
SELECT BIT_LENGTH(null::binary);	
SELECT BIT_LENGTH(VARCHAR 'abc');	24
SELECT BIT_LENGTH(CHAR 'abc');	24
SELECT BIT_LENGTH(CHAR(6) 'abc');	48
SELECT BIT_LENGTH(VARCHAR(6) 'abc');	24
SELECT BIT_LENGTH(BINARY(6) 'abc');	48
SELECT BIT_LENGTH(BINARY 'abc');	24

SELECT BIT_LENGTH(VARBINARY 'abc');	24
SELECT BIT_LENGTH(VARBINARY(6) 'abc');	24

See Also

- [CHARACTER_LENGTH](#)
- [LENGTH](#)
- [OCTET_LENGTH](#)

BITCOUNT

Returns the number of one-bits (sometimes referred to as set-bits) in the given VARBINARY value. This is also referred to as the population count.

Behavior Type

Immutable

Syntax

```
BITCOUNT ( expression )
```

Parameters

<i>expression</i>	(BINARY or VARBINARY) is the string to return.
-------------------	--

Examples

```
=> SELECT BITCOUNT(HEX_TO_BINARY('0x10'));
   BITCOUNT
-----
           1
(1 row)
```

```
=> SELECT BITCOUNT(HEX_TO_BINARY('0xF0'));
   BITCOUNT
-----
          4
(1 row)

=> SELECT BITCOUNT(HEX_TO_BINARY('0xAB'));
   BITCOUNT
-----
          5
(1 row)
```

BITSTRING_TO_BINARY

Translates the given VARCHAR bitstring representation into a VARBINARY value. This function is the inverse of [TO_BITSTRING](#).

Behavior Type

Immutable

Syntax

```
BITSTRING_TO_BINARY ( expression )
```

Parameters

<i>expression</i>	The VARCHAR string to process.
-------------------	--------------------------------

Examples

If there are an odd number of characters in the hex value, the first character is treated as the low nibble of the first (furthest to the left) byte.

```
=> SELECT BITSTRING_TO_BINARY('0110000101100010');
   BITSTRING_TO_BINARY
-----
          ab
(1 row)
```

BTRIM

Removes the longest string consisting only of specified characters from the start and end of a string.

Behavior Type

Immutable

Syntax

```
BTRIM ( expression [ , characters-to-remove ] )
```

Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to modify
<i>characters-to-remove</i>	(CHAR or VARCHAR) specifies the characters to remove. The default is the space character.

Example

```
=> SELECT BTRIM('yxtrimyx', 'xy');  
BTRIM  
-----  
trim  
(1 row)
```

See Also

- [LTRIM](#)
- [RTRIM](#)
- [TRIM](#)

CHARACTER_LENGTH

The CHARACTER_LENGTH() function:

- Returns the string length in UTF-8 characters for CHAR and VARCHAR columns
- Returns the string length in bytes (octets) for BINARY and VARBINARY columns
- Strips the padding from CHAR expressions but not from VARCHAR expressions
- Is identical to [LENGTH\(\)](#) for CHAR and VARCHAR. For binary types, CHARACTER_LENGTH() is identical to [OCTET_LENGTH\(\)](#).

Behavior Type

Immutable if USING OCTETS, stable otherwise.

Syntax

```
[ CHAR_LENGTH | CHARACTER_LENGTH ] ( expression ... [ USING { CHARACTERS | OCTETS } ] )
```

Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to measure
USING CHARACTERS OCTETS	Determines whether the character length is expressed in characters (the default) or octets.

Examples

```
=> SELECT CHAR_LENGTH('1234 '::CHAR(10) USING OCTETS);
   octet_length
-----
           4
(1 row)

=> SELECT CHAR_LENGTH('1234 '::VARCHAR(10));
   char_length
-----
```

```
        6  
(1 row)  
=> SELECT CHAR_LENGTH(NULL::CHAR(10)) IS NULL;  
?column?  
-----  
t  
(1 row)
```

See Also

- [BIT_LENGTH](#)

CHR

Converts the first character of an INTEGER datatype to a VARCHAR.

Behavior Type

Immutable

Syntax

CHR (*expression*)

Parameters

<i>expression</i>	(INTEGER) is the string to convert and is masked to a single character.
-------------------	---

Notes

- CHR is the opposite of the [ASCII](#) function.
- CHR operates on UTF-8 characters, not only on single-byte ASCII characters. It continues to get the same results for the ASCII subset of UTF-8.

Examples

This example returns the VARCHAR datatype of the CHR expressions 65 and 97 from the employee table:

```
=> SELECT CHR(65), CHR(97) FROM employee;
CHR | CHR
-----+-----
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
A   | a
(12 rows)
```

COLLATION

Applies a collation to two or more strings. Use COLLATION with ORDER BY, GROUP BY, and equality clauses.

Syntax

```
COLLATION ( 'expression' [ , 'locale_or_collation_name' ] )
```

Parameters

'expression'	Any expression that evaluates to a column name or to two or more values of type CHAR or VARCHAR.
'locale_or_collation_name'	The ICU (International Components for Unicode) locale or collation name to use when collating the string. If you omit this parameter, COLLATION uses the collation associated with the session locale. To determine the current session locale, enter the vsql meta-command <code>\locale</code> :

```
=> \locale
en_US@collation=binary
```

To set the locale and collation, use `\locale` as follows:

```
=> \locale en_US@collation=binary
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
```

Locales

The locale used for COLLATION can be one of the following:

- The default locale
- A session locale
- A locale that you specify when you call COLLATION. If you specify the locale, Vertica applies the collation associated with that locale to the data. COLLATION does not modify the collation for any other columns in the table.

For a list of valid ICU locales, go to [Locale Explorer \(ICU\)](#).

Binary and Non-Binary Collations

The Vertica default locale is `en_US@collation=binary`, which uses *binary collation*. Binary collation compares binary representations of strings. Binary collation is fast, but it can result in a sort order where K precedes c because the binary representation of K is lower than c.

For non-binary collation, Vertica transforms the data according to the rules of the locale or the specified collation, and then applies the sorting rules. Suppose the locale collation is non-binary and you request a GROUP BY on string data. In this case, Vertica calls COLLATION, whether or not you specify the function in your query.

For information about collation naming, see [Collator Naming Scheme](#).

Examples

Collating GROUP BY Results

The following examples are based on a `Premium_Customer` table that contains the following data:

```
=> SELECT * FROM Premium_Customer;
ID | LName | FName
----+-----+-----
 1 | Mc Coy | Bob
 2 | Mc Coy | Janice
 3 | McCoy | Jody
 4 | McCoy | Peter
 5 | McCoy | Brendon
 6 | Mccoy | Cameron
 7 | Mccoy | Lisa
```

The first statement shows how COLLATION applies the collation for the EN_US locale to the LName column for the locale EN_US. Vertica sorts the GROUP BY output as follows:

- Last names with spaces
- Last names where "coy" starts with a lowercase letter
- Last names where "Coy" starts with an uppercase letter

```
=> SELECT * FROM Premium_Customer ORDER BY COLLATION(LName, 'EN_US'), FName;
ID | LName | FName
----+-----+-----
 1 | Mc Coy | Bob
 2 | Mc Coy | Janice
 6 | Mccoy | Cameron
 7 | Mccoy | Lisa
 5 | McCoy | Brendon
 3 | McCoy | Jody
 4 | McCoy | Peter
```

The next statement shows how COLLATION collates the LName column for the locale LEN_AS:

- LEN indicates the language (L) is English (EN).
- AS (Alternate Shifted) instructs COLLATION that lowercase letters come before uppercase (shifted) letters.

In the results, the last names in which "coy" starts with a lowercase letter precede the last names where "Coy" starts with an uppercase letter.

```
=> SELECT * FROM Premium_Customer ORDER BY COLLATION(LName, 'LEN_AS'), FName;
ID | LName | FName
----+-----+-----
 6 | Mccoy | Cameron
 7 | Mccoy | Lisa
 1 | Mc Coy | Bob
 5 | McCoy | Brendon
 2 | Mc Coy | Janice
 3 | McCoy | Jody
 4 | McCoy | Peter
```

Comparing Strings with an Equality Clause

In the following query, COLLATION removes spaces and punctuation when comparing two strings in English. It then determines whether the two strings still have the same value after the punctuation has been removed:

```
=> SELECT COLLATION ('U.S.A', 'LEN_AS') = COLLATION('USA', 'LEN_AS');
?column?
-----
t
```

Sorting Strings in Non-English Languages

The following table contains data that uses the German character eszett, ß:

```
=> SELECT * FROM t1;
   a      | b | c
-----+-----+-----
ßstringß | 1 | 10
SSstringSS | 2 | 20
random1   | 3 | 30
random1   | 4 | 40
random2   | 5 | 50
```

When you specify the collation LDE_S1:

- LDE indicates the language (L) is German (DE).
- S1 indicates the strength (S) of 1 (primary). This value indicates that the collation does not need to consider accents and case.

The query returns the data in the following order:

```
=> SELECT a FROM t1 ORDER BY COLLATION(a, 'LDE_S1');
a
-----
random1
random1
random2
SSstringSS
ßstringß
```

CONCAT

Used to concatenate two strings.

Syntax

```
CONCAT ('string1','string2')
```

Behavior Type

Immutable

Parameters

'string1'	Can be any datatype.
'string2'	

Restrictions

Varbinary and long varbinary types cannot be mixed with other types. These types return varbinary and long varbinary, respectively.

Similarly, long varchar types return long varchar.

Otherwise, the result is varchar.

Note: If either argument is null, concat returns null.

Example

The following simple examples use a sample table named `alphabet`, which contains two rows, `letter1` and `letter2`. The contents are as follows.

```
=> CREATE TABLE alphabet (letter1 varchar(2), letter2 varchar(2));
CREATE TABLE
=> COPY alphabet FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> A|B
>> C|D
>> \.
=> SELECT * FROM alphabet;
 letter1 | letter2
-----+-----
 C      | D
 A      | B
(2 rows)
```

The following example concatenates the contents of the first column with a character string.

```
=> SELECT CONCAT(letter1, ' is a letter') FROM alphabet;
      CONCAT
-----
A is a letter
C is a letter
(2 rows)
```

The following example nests the CONCAT function.

```
=> SELECT CONCAT(CONCAT(letter1, ' and '), CONCAT(letter2, ' are both letters')) FROM alphabet;
      CONCAT
-----
C and D are both letters
A and B are both letters
(2 rows)
```

DECODE

Compares *expression* to each search value one by one. If *expression* is equal to a search, the function returns the corresponding result. If no match is found, the function returns default. If default is omitted, the function returns null.

DECODE is similar to the IF-THEN-ELSE and [CASE](#) expression:

```
CASE expression WHEN search THEN result
[WHEN search THEN result]
[ELSE default];
```

The arguments can have any data type supported by Vertica. The result types of individual results are promoted to the least common type that can be used to represent all of them. This leads to a character string type, an exact numeric type, an approximate numeric type, or a DATETIME type, where all the various result arguments must be of the same type grouping.

Behavior Type

Immutable

Syntax

```
DECODE ( expression, search, result [ , search, result ]...[, default ] )
```

Parameters

<i>expression</i>	The value to compare.
<i>search</i>	The value compared against <i>expression</i> .
<i>result</i>	The value returned, if <i>expression</i> is equal to search.
<i>default</i>	Optional. If no matches are found, DECODE returns default. If default is omitted, then DECODE returns NULL (if no matches are found).

Example

The following example converts numeric values in the weight column from the product_ dimension table to descriptive values in the output.

```
=> SELECT product_description, DECODE(weight,
    2, 'Light',
    50, 'Medium',
    71, 'Heavy',
    99, 'Call for help',
    'N/A')
FROM product_dimension
WHERE category_description = 'Food'
AND department_description = 'Canned Goods'
AND sku_number BETWEEN 'SKU-#49750' AND 'SKU-#49999'
LIMIT 15;
```

product_description	case
Brand #499 canned corn	N/A
Brand #49900 fruit cocktail	Medium
Brand #49837 canned tomatoes	Heavy
Brand #49782 canned peaches	N/A
Brand #49805 chicken noodle soup	N/A
Brand #49944 canned chicken broth	N/A
Brand #49819 canned chili	N/A
Brand #49848 baked beans	N/A
Brand #49989 minestrone soup	N/A
Brand #49778 canned peaches	N/A
Brand #49770 canned peaches	N/A
Brand #4977 fruit cocktail	N/A
Brand #49933 canned olives	N/A
Brand #49750 canned olives	Call for help
Brand #49777 canned tomatoes	N/A

(15 rows)

GREATEST

Returns the largest value in a list of expressions.

Behavior Type

Stable

Syntax

```
GREATEST ( expression1, expression2, ... expression-n )
```

Parameters

expression1, *expression2*, and *expression-n* are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples.
- A NULL value in any one of the expressions returns NULL.
- Depends on the collation setting of the locale.

Examples

This example returns 9 as the greatest in the list of expressions:

```
=> SELECT GREATEST(7, 5, 9);
GREATEST
-----
          9
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:

```
=> SELECT GREATEST('7', '5', '9');
GREATEST
-----
9
(1 row)
```

The next example returns FLOAT 1.5 as the greatest because the integer is implicitly cast to float:

```
=> SELECT GREATEST(1, 1.5);
GREATEST
-----
1.5
(1 row)
```

The following example returns 'vertica' as the greatest:

```
=> SELECT GREATEST('vertica', 'analytic', 'database');
GREATEST
-----
vertica
(1 row)
```

Notice this next command returns NULL:

```
=> SELECT GREATEST('vertica', 'analytic', 'database', null);
GREATEST
-----
(1 row)
```

And one more:

```
=> SELECT GREATEST('sit', 'site', 'sight');
GREATEST
-----
site
(1 row)
```

See Also

- [LEAST](#)

GREATESTB

Returns its greatest argument, using binary ordering, not UTF-8 character ordering.

Behavior Type

Immutable

Syntax

```
GREATESTB ( expression1, expression2, ... expression-n )
```

Parameters

expression1, *expression2*, and *expression-n* are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples.
- A NULL value in any one of the expressions returns NULL.
- Depends on the collation setting of the locale.

Examples

The following command selects *straße* as the greatest in the series of inputs:

```
=> SELECT GREATESTB('straße', 'strasse');
GREATESTB
-----
straße
(1 row)
```

This example returns 9 as the greatest in the list of expressions:

```
=> SELECT GREATESTB(7, 5, 9);
GREATESTB
-----
          9
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:

```
=> SELECT GREATESTB('7', '5', '9');
GREATESTB
-----
9
(1 row)
```

The next example returns FLOAT 1.5 as the greatest because the integer is implicitly cast to float:

```
=> SELECT GREATESTB(1, 1.5);
GREATESTB
-----
1.5
(1 row)
```

The following example returns `vertica` as the greatest:

```
=> SELECT GREATESTB('vertica', 'analytic', 'database');
GREATESTB
-----
vertica
(1 row)
```

Notice this next command returns NULL:

```
=> SELECT GREATESTB('vertica', 'analytic', 'database', null);
GREATESTB
-----
(1 row)
```

And one more:

```
=> SELECT GREATESTB('sit', 'site', 'sight');
GREATESTB
-----
site
(1 row)
```

See Also

- [LEASTB](#)

HEX_TO_BINARY

Translates the given VARCHAR hexadecimal representation into a VARBINARY value.

Behavior Type

Immutable

Syntax

```
HEX_TO_BINARY ( [ 0x ] expression )
```

Parameters

<i>expression</i>	(BINARY or VARBINARY) String to translate.
0x	Optional prefix.

Notes

VARBINARY HEX_TO_BINARY(VARCHAR) converts data from character type in hexadecimal format to binary type. This function is the inverse of [TO_HEX](#).

```
HEX_TO_BINARY(TO_HEX(x)) = x  
TO_HEX(HEX_TO_BINARY(x)) = x
```

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low nibble of the first (furthest to the left) byte.

Examples

If the given string begins with "0x" the prefix is ignored. For example:

```
=> SELECT HEX_TO_BINARY('0x6162') AS hex1, HEX_TO_BINARY('6162') AS hex2;  
hex1 | hex2  
-----+-----  
ab   | ab  
(1 row)
```

If an invalid hex value is given, Vertica returns an "invalid binary representation" error; for example:

```
=> SELECT HEX_TO_BINARY('0xffgf');  
ERROR:  invalid hex string "0xffgf"
```

See Also

- [TO_HEX](#)

HEX_TO_INTEGER

Translates the given VARCHAR hexadecimal representation into an INTEGER value.

Vertica completes this conversion as follows:

- Adds the 0x prefix if it is not specified in the input
- Casts the VARCHAR string to a NUMERIC
- Casts the NUMERIC to an INTEGER

Behavior Type

Immutable

Syntax

```
HEX_TO_INTEGER ( [ 0x ] expression )
```

Parameters

<i>expression</i>	VARCHAR is the string to translate.
0x	Is the optional prefix.

Examples

You can enter the string with or without the 0x prefix. For example:

```
=> SELECT HEX_TO_INTEGER ('0aedic')
      AS hex1,HEX_TO_INTEGER ('aedic') AS hex2;
hex1  | hex2
-----+-----
44764 | 44764
(1 row)
```

If you pass the function an invalid hex value, Vertica returns an `invalid input syntax` error; for example:

```
=> SELECT HEX_TO_INTEGER ('0xffgf');
ERROR 3691: Invalid input syntax for numeric: "0xffgf"
```

See Also

- [TO_HEX](#)
- [HEX_TO_BINARY](#)

INET_ATON

Returns an integer that represents the value of the address in host byte order, given the dotted-quad representation of a network address as a string.

Behavior Type

Immutable

Syntax

```
INET_ATON ( expression )
```

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Notes

The following syntax converts an IPv4 address represented as the string *A* to an integer *I*. `INET_ATON` trims any spaces from the right of *A*, calls the Linux function `inet_pton`, and converts the result from network byte order to host byte order using `ntohl`.

```
=> INET_ATON(VARCHAR A) -> INT8 I
```

If *A* is NULL, too long, or `inet_pton` returns an error, the result is NULL.

Examples

The generated number is always in host byte order. In the following example, the number is calculated as $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$.

```
=> SELECT INET_ATON('209.207.224.40');
inet_aton
-----
3520061480
(1 row)

=> SELECT INET_ATON('1.2.3.4');
inet_aton
-----
16909060
(1 row)

=> SELECT TO_HEX(INET_ATON('1.2.3.4'));
to_hex
-----
1020304
(1 row)
```

See Also

- [INET_NTOA](#)

INET_NTOA

Returns the dotted-quad representation of the address as a VARCHAR, given a network address as an integer in network byte order.

Behavior Type

Immutable

Syntax

```
INET_NTOA ( expression )
```

Parameters

<i>expression</i>	(INTEGER) is the network address to convert.
-------------------	--

Notes

The following syntax converts an IPv4 address represented as integer I to a string A.

INET_NTOA converts I from host byte order to network byte order using [htonl](#), and calls the Linux function [inet_ntop](#).

```
=> INET_NTOA(INT8 I) -> VARCHAR A
```

If I is NULL, greater than 2^{32} or negative, the result is NULL.

Examples

```
=> SELECT INET_NTOA(16909060);
inet_ntoa
-----
1.2.3.4
(1 row)

=> SELECT INET_NTOA(03021962);
inet_ntoa
-----
0.46.28.138
(1 row)
```

See Also

- [INET_ATON](#)

INITCAP

Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.

Behavior Type

Immutable

Syntax

INITCAP (*expression*)

Parameters

<i>expression</i>	(VARCHAR) is the string to format.
-------------------	------------------------------------

Notes

- Depends on collation setting of the locale.
- INITCAP is restricted to 32750 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

Examples

Expression	Result
SELECT INITCAP('high speed database');	High Speed Database
SELECT INITCAP('LINUX TUTORIAL');	Linux Tutorial

SELECT INITCAP('abc DEF 123aVc 124Btd,1AsT');	Abc Def 123Avc 124Btd,Last
SELECT INITCAP('');	
SELECT INITCAP(null);	

INITCAPB

Capitalizes first letter of each alphanumeric word and puts the rest in lowercase. Multibyte characters are not converted and are skipped.

Behavior Type

Immutable

Syntax

INITCAPB (*expression*)

Parameters

<i>expression</i>	(VARCHAR) is the string to format.
-------------------	------------------------------------

Notes

Depends on collation setting of the locale.

Examples

Expression	Result
SELECT INITCAPB('étudiant');	éTudiant
SELECT INITCAPB('high speed database');	High Speed Database
SELECT INITCAPB('LINUX TUTORIAL');	Linux Tutorial
SELECT INITCAPB('abc DEF 123aVc 124Btd,1AsT');	Abc Def 123Avc 124Btd,Last

SELECT INITCAPB('');	
SELECT INITCAPB(null);	

INSERT

Inserts a character string into a specified location in another character string.

Syntax

```
INSERT( 'string1', n, m, 'string2' )
```

Behavior Type

Immutable

Parameters

<i>string1</i>	(VARCHAR) Is the string in which to insert the new string.
<i>n</i>	A character of type INTEGER that represents the starting point for the insertion within <i>string1</i> . You specify the number of characters from the first character in <i>string1</i> as the starting point for the insertion. For example, to insert characters before "c", in the string "abcdef," enter 3.
<i>m</i>	A character of type INTEGER that represents the number of characters in <i>string1</i> (if any) that should be replaced by the insertion. For example, if you want the insertion to replace the letters "cd" in the string "abcdef, " enter 2.
<i>string2</i>	(VARCHAR) Is the string to be inserted.

Example

The following example changes the string Warehouse to Storehouse using the INSERT function:

```
=> SELECT INSERT ('Warehouse',1,3,'Stor');  
      INSERT  
-----
```

Storehouse
(1 row)

INSTR

Searches *string* for *substring* and returns an integer indicating the position of the character in *string* that is the first character of this *occurrence*. The return value is based on the character position of the identified character.

Behavior Type

Immutable

Syntax

```
INSTR ( string , substring [, position [, occurrence ] ] )
```

Parameters

<i>string</i>	(CHAR or VARCHAR, or BINARY or VARBINARY) Text expression to search.
<i>substring</i>	(CHAR or VARCHAR, or BINARY or VARBINARY) String to search for.
<i>position</i>	Nonzero integer indicating the character of string where Vertica begins the search. If position is negative, then Vertica counts backward from the end of string and then searches backward from the resulting position. The first character of string occupies the default position 1, and position cannot be 0.
<i>occurrence</i>	Integer indicating which occurrence of string Vertica searches. The value of occurrence must be positive (greater than 0), and the default is 1.

Notes

Both *position* and *occurrence* must be of types that can resolve to an integer. The default values of both parameters are 1, meaning Vertica begins searching at the first character of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in characters.

If the search is unsuccessful (that is, if substring does not appear *occurrence* times after the *position* character of *string*, the return value is 0.

Examples

The first example searches forward in string 'abc' for substring 'b'. The search returns the position in 'abc' where 'b' occurs, or position 2. Because no position parameters are given, the default search starts at 'a', position 1.

```
=> SELECT INSTR('abc', 'b');
INSTR
-----
      2
(1 row)
```

The following three examples use character position to search backward to find the position of a substring.

Note: Although it might seem intuitive that the function returns a negative integer, the position of *n* occurrence is read left to right in the sting, even though the search happens in reverse (from the end—or right side—of the string).

In the first example, the function counts backward one character from the end of the string, starting with character 'c'. The function then searches backward for the first occurrence of 'a', which it finds it in the first position in the search string.

```
=> SELECT INSTR('abc', 'a', -1);
INSTR
-----
      1
(1 row)
```

In the second example, the function counts backward one byte from the end of the string, starting with character 'c'. The function then searches backward for the first occurrence of 'a', which it finds it in the first position in the search string.

```
=> SELECT INSTR(VARBINARY 'abc', VARBINARY 'a', -1);
INSTR
-----
      1
(1 row)
```

In the third example, the function counts backward one character from the end of the string, starting with character 'b', and searches backward for substring 'bc', which it finds in the second position of the search string.

```
=> SELECT INSTR('abcb', 'bc', -1);
INSTR
-----
      2
(1 row)
```

In the fourth example, the function counts backward one character from the end of the string, starting with character 'b', and searches backward for substring 'bcef', which it does not find. The result is 0.

```
=> SELECT INSTR('abcb', 'bcef', -1);
INSTR
-----
      0
(1 row)
```

In the fifth example, the function counts backward one byte from the end of the string, starting with character 'b', and searches backward for substring 'bcef', which it does not find. The result is 0.

```
=> SELECT INSTR(VARBINARY 'abcb', VARBINARY 'bcef', -1);
INSTR
-----
      0
(1 row)
```

Multibyte characters are treated as a single character:

```
=> SELECT INSTR('aébc', 'b');
INSTR
-----
      3
(1 row)
```

Use INSTRB to treat multibyte characters as binary:

```
=> SELECT INSTRB('aébc', 'b');
INSTRB
-----
      4
(1 row)
```

INSTRB

Searches *string* for *substring* and returns an integer indicating the octet position within string that is the first *occurrence*. The return value is based on the octet position of the identified byte.

Behavior Type

Immutable

Syntax

```
INSTRB ( string , substring [, position [, occurrence ] ] )
```

Parameters

<i>string</i>	Is the text expression to search.
<i>substring</i>	Is the string to search for.
<i>position</i>	Is a nonzero integer indicating the character of string where Vertica begins the search. If position is negative, then Vertica counts backward from the end of string and then searches backward from the resulting position. The first byte of string occupies the default position 1, and position cannot be 0.
<i>occurrence</i>	Is an integer indicating which occurrence of string Vertica searches. The value of occurrence must be positive (greater than 0), and the default is 1.

Notes

Both *position* and *occurrence* must be of types that can resolve to an integer. The default values of both parameters are 1, meaning Vertica begins searching at the first byte of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in octets.

If the search is unsuccessful (that is, if substring does not appear *occurrence* times after the *position* character of *string*, then the return value is 0.

Example

```
=> SELECT INSTRB('straße', 'ß');  
INSTRB  
-----  
5
```

(1 row)

See Also

- [INSTR](#)

ISUTF8

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as [REGEXP_LIKE](#), which expect UTF-8 characters by default.

ISUTF8 checks for invalid UTF8 byte sequences, according to UTF-8 rules:

- invalid bytes
- an unexpected continuation byte
- a start byte not followed by enough continuation bytes
- an Overload Encoding

The presence of an invalid UTF8 byte sequence results in a return value of false.

Syntax

```
ISUTF8( string );
```

Parameters

<i>string</i>	The string to test for UTF-8 compliance.
---------------	--

Examples

```
=> SELECT ISUTF8(E'\xC2\xBF'); -- UTF-8 INVERTED QUESTION MARK ISUTF8  
-----
```

```
t
(1 row)
=> SELECT ISUTF8(E'\xC2\xC0'); -- UNDEFINED UTF-8 CHARACTER
ISUTF8
-----
f
(1 row)
```

LEAST

Returns the smallest value in a list of expressions.

Behavior Type

Stable

Syntax

```
LEAST ( expression1, expression2, ... expression-n )
```

Parameters

expression1, *expression2*, and *expression-n* are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples below.
- A NULL value in any one of the expressions returns NULL.

Examples

This example returns 5 as the least:

```
=> SELECT LEAST(7, 5, 9);
LEAST
-----
5
(1 row)
```

Putting quotes around the integer expressions returns the same result as the first example:

```
=> SELECT LEAST('7', '5', '9');
LEAST
-----
5
(1 row)
```

In the above example, the values are being compared as strings, so '10' would be less than '2'.

The next example returns 1.5, as INTEGER 2 is implicitly cast to FLOAT:

```
=> SELECT LEAST(2, 1.5);
LEAST
-----
1.5
(1 row)
```

The following example returns 'analytic' as the least:

```
=> SELECT LEAST('vertica', 'analytic', 'database');
LEAST
-----
analytic
(1 row)
```

Notice this next command returns NULL:

```
=> SELECT LEAST('vertica', 'analytic', 'database', null);
LEAST
-----

(1 row)
```

And one more:

```
=> SELECT LEAST('sit', 'site', 'sight');
LEAST
-----
sight
(1 row)
```

See Also

- [GREATEST](#)

LEASTB

Returns the function's least argument, using binary ordering, not UTF-8 character ordering.

Behavior Type

Immutable

Syntax

```
LEASTB ( expression1, expression2, ... expression-n )
```

Parameters

expression1, *expression2*, and *expression-n* are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples below.
- A NULL value in any one of the expressions returns NULL.

Examples

The following command selects *strasse* as the least in the series of inputs:

```
=> SELECT LEASTB('straße', 'strasse');
LEASTB
-----
strasse
(1 row)
```

This example returns 5 as the least:

```
==> SELECT LEASTB(7, 5, 9);
LEASTB
-----
5
(1 row)
```

Putting quotes around the integer expressions returns the same result as the first example:

```
=> SELECT LEASTB('7', '5', '9');
LEASTB
-----
5
```

```
(1 row)
```

In the above example, the values are being compared as strings, so '10' would be less than '2'.

The next example returns 1.5, as INTEGER 2 is implicitly cast to FLOAT:

```
=> SELECT LEASTB(2, 1.5);
LEASTB
-----
      1.5
(1 row)
```

The following example returns 'analytic' as the least in the series of inputs:

```
=> SELECT LEASTB('vertica', 'analytic', 'database');
LEASTB
-----
analytic
(1 row)
```

Notice this next command returns NULL:

```
=> SELECT LEASTB('vertica', 'analytic', 'database', null);
LEASTB
-----

(1 row)
```

See Also

- [GREATESTB](#)

LEFT

Returns the specified characters from the left side of a string.

Behavior Type

Immutable

Syntax

```
LEFT ( string , Length )
```

Parameters

<i>string</i>	(CHAR or VARCHAR) is the string to return.
<i>length</i>	Is an INTEGER value that specifies the count of characters to return.

Examples

```
=> SELECT LEFT('vertica', 3);
LEFT
-----
ver
(1 row)

=> SELECT LEFT('straße', 5);
LEFT
-----
straß
(1 row)
```

See Also

- [SUBSTR](#)

LENGTH

Returns the length of a string. The behavior of LENGTH varies according to the input data type:

- CHAR and VARCHAR: Identical to [CHARACTER_LENGTH](#), returns the string length in UTF-8 characters, .
- CHAR: Strips padding.
- BINARY and VARBINARY: Identical to [OCTET_LENGTH](#), returns the string length in bytes (octets).

Behavior Type

Immutable

Syntax

LENGTH (*expression*)

Parameters

<i>expression</i>	String to evaluate, one of the following: CHAR, VARCHAR, BINARY or VARBINARY.
-------------------	---

Examples

Statement	Returns
SELECT LENGTH('1234 '::CHAR(10));	4
SELECT LENGTH('1234 '::VARCHAR(10));	6
SELECT LENGTH('1234 '::BINARY(10));	10
SELECT LENGTH('1234 '::VARBINARY(10));	6
SELECT LENGTH(NULL::CHAR(10)) IS NULL;	t

See Also

[BIT_LENGTH](#)

LOWER

Returns a VARCHAR value containing the argument converted to lowercase letters.

LOWER treats the `string` argument as a UTF-8 encoded string, rather than depending on the collation setting of the locale (for example, `collation=binary`) to identify the encoding.

Behavior Type

stable

Syntax

`LOWER (expression)`

Parameters

<i>expression</i>	CHAR or VARCHAR string to convert
-------------------	-----------------------------------

Notes

LOWER is restricted to 32500 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

Examples

```
=> SELECT LOWER('AbCdEfg');
      LOWER
-----
abcdefg
(1 row)

=> SELECT LOWER('The Bat In The Hat');
      LOWER
-----
the bat in the hat
(1 row)

=> SELECT LOWER('ÉTUDIANT');
      LOWER
-----
étudiant
(1 row)
```

LOWERRB

Returns a character string with each ASCII character converted to lowercase. Multi-byte characters are skipped and not converted.

Behavior Type

Immutable

Syntax

`LOWERB (expression)`

Parameters

<i>expression</i>	CHAR or VARCHAR string to convert
-------------------	-----------------------------------

Examples

In the following example, the multi-byte UTF-8 character É is not converted to lowercase:

```
=> SELECT LOWERB('ÉTUDIANT');
      LOWERB
-----
Étudiant
(1 row)

=> SELECT LOWER('ÉTUDIANT');
      LOWER
-----
étudiant
(1 row)

=> SELECT LOWERB('AbCdEfg');
      LOWERB
-----
abcdefg
(1 row)

=> SELECT LOWERB('The Vertica Database');
      LOWERB
-----
the vertica database
(1 row)
```

LPAD

Returns a VARCHAR value representing a string of a specific length filled on the left with specific characters.

Behavior Type

Immutable

Syntax

`LPAD (expression , Length [, fill])`

Parameters

<i>expression</i>	(CHAR OR VARCHAR) specifies the string to fill
<i>Length</i>	(INTEGER) specifies the number of characters to return
<i>fill</i>	(CHAR OR VARCHAR) specifies the repeating string of characters with which to fill the output string. The default is the space character.

Examples

```
=> SELECT LPAD('database', 15, 'xzy');
      LPAD
-----
 xzyxzyxdatabase
(1 row)
```

If the string is already longer than the specified length it is truncated on the right:

```
=> SELECT LPAD('establishment', 10, 'abc');
      LPAD
-----
 establishm
(1 row)
```

LTRIM

Returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).

Behavior Type

Immutable

Syntax

`LTRIM (expression [, characters])`

Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to trim
<i>characters</i>	(CHAR or VARCHAR) specifies the characters to remove from the left side of <i>expression</i> . The default is the space character.

Examples

```
=> SELECT LTRIM('zzzyyyyyxxxxxxxxtrim', 'xyz');
   LTRIM
-----
trim
(1 row)
```

See Also

- [BTRIM](#)
- [RTRIM](#)
- [TRIM](#)

MD5

Calculates the MD5 hash of string, returning the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

MD5 (*string*)

Parameters

<i>string</i>	Is the argument string.
---------------	-------------------------

Examples

```
=> SELECT MD5('123');
      MD5
-----
202cb962ac59075b964b07152d234b70
(1 row)

=> SELECT MD5('Vertica'::bytea);
      MD5
-----
fc45b815747d8236f9f6fdb9c2c3f676
(1 row)
```

See Also

- [SHA1](#)
- [SHA224](#)
- [SHA256](#)
- [SHA384](#)
- [SHA512](#)

OCTET_LENGTH

Takes one argument as an input and returns the string length in octets for all string types.

Behavior Type

Immutable

Syntax

```
OCTET_LENGTH ( expression )
```

Parameters

<i>expression</i>	(CHAR or VARCHAR or BINARY or VARBINARY) is the string to measure.
-------------------	--

Notes

- If the data type of *expression* is a CHAR, VARCHAR or VARBINARY, the result is the same as the actual length of *expression* in octets. For CHAR, the length does not include any trailing spaces.
- If the data type of *expression* is BINARY, the result is the same as the fixed-length of *expression*.
- If the value of *expression* is NULL, the result is NULL.

Examples

Expression	Result
SELECT OCTET_LENGTH(CHAR(10) '1234 ');	4
SELECT OCTET_LENGTH(CHAR(10) '1234');	4
SELECT OCTET_LENGTH(CHAR(10) ' 1234');	6
SELECT OCTET_LENGTH(VARCHAR(10) '1234 ');	6
SELECT OCTET_LENGTH(VARCHAR(10) '1234');	5
SELECT OCTET_LENGTH(VARCHAR(10) '1234');	4
SELECT OCTET_LENGTH(VARCHAR(10) ' 1234');	7

SELECT OCTET_LENGTH('abc'::VARBINARY);	3
SELECT OCTET_LENGTH(VARBINARY 'abc');	3
SELECT OCTET_LENGTH(VARBINARY 'abc ');	5
SELECT OCTET_LENGTH(BINARY(6) 'abc');	6
SELECT OCTET_LENGTH(VARBINARY '');	0
SELECT OCTET_LENGTH('':BINARY);	1
SELECT OCTET_LENGTH(null::VARBINARY);	
SELECT OCTET_LENGTH(null::BINARY);	

See Also

- [BIT_LENGTH](#)
- [CHARACTER_LENGTH](#)
- [LENGTH](#)

OVERLAY

Returns a VARCHAR value representing a string having had a substring replaced by another string.

Behavior Type

Immutable if using OCTETS, Stable otherwise

Syntax

```
OVERLAY ( expression1 PLACING expression2 FROM position  
... [ FOR extent ]  
... [ USING { CHARACTERS | OCTETS } ] )
```

Parameters

<i>expression1</i>	(CHAR or VARCHAR) is the string to process
<i>expression2</i>	(CHAR or VARCHAR) is the substring to overlay
<i>position</i>	(INTEGER) is the character or octet position (counting from one) at which to begin the overlay
<i>extent</i>	(INTEGER) specifies the number of characters or octets to replace with the overlay
USING CHARACTERS OCTETS	Determines whether OVERLAY uses characters (the default) or octets

Examples

```
=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2);
  overlay
-----
1xxx56789
(1 row)

=> SELECT OVERLAY('123456789' PLACING 'XXX' FROM 2 USING OCTETS);
  overlayb
-----
1XXX56789
(1 row)

=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 4);
  overlay
-----
1xxx6789
(1 row)

=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 5);
  overlay
-----
1xxx789
(1 row)

=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 6);
  overlay
-----
1xxx89
(1 row)
```

OVERLAYB

Returns an octet value representing a string having had a substring replaced by another string.

Behavior Type

Immutable

Syntax

```
OVERLAYB ( expression1, expression2, position [ , extent ] )
```

Parameters

<i>expression1</i>	(CHAR or VARCHAR) is the string to process
<i>expression2</i>	(CHAR or VARCHAR) is the substring to overlay
<i>position</i>	(INTEGER) is the octet position (counting from one) at which to begin the overlay
<i>extent</i>	(INTEGER) specifies the number of octets to replace with the overlay

Notes

The OVERLAYB function treats the multibyte character string as a string of octets (bytes) and use octet numbers as incoming and outgoing position specifiers and lengths. The strings themselves are type VARCHAR, but they treated as if each byte was a separate character.

Examples

```
=> SELECT OVERLAYB('123456789', 'ééé', 2);
OVERLAYB
-----
1ééé89
(1 row)
```

```
=> SELECT OVERLAYB('123456789', 'BBB', 2);
OVERLAYB
-----
1BBB89
(1 row)

=> SELECT OVERLAYB('123456789', 'xxx', 2);
OVERLAYB
-----
1xxx56789
(1 row)

=> SELECT OVERLAYB('123456789', 'xxx', 2, 4);
OVERLAYB
-----
1xxx6789
(1 row)

=> SELECT OVERLAYB('123456789', 'xxx', 2, 5);
OVERLAYB
-----
1xxx789
(1 row)

=> SELECT OVERLAYB('123456789', 'xxx', 2, 6);
OVERLAYB
-----
1xxx89
(1 row)
```

POSITION

Returns an INTEGER value representing the character location of a specified substring with a string (counting from one).

Behavior Type

Immutable

Syntax 1

```
POSITION ( substring IN string [ USING { CHARACTERS | OCTETS } ] )
```

Parameters

<i>substring</i>	(CHAR or VARCHAR) is the substring to locate
------------------	--

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
USING CHARACTERS OCTETS	Determines whether the position is reported by using characters (the default) or octets.

Syntax 2

POSITION (*substring* IN *string*)

Parameters

<i>substring</i>	(VARBINARY) is the substring to locate
<i>string</i>	(VARBINARY) is the string in which to locate the substring

Notes

- When the string and substring are CHAR or VARCHAR, the return value is based on either the character or octet position of the substring.
- When the string and substring are VARBINARY, the return value is always based on the octet position of the substring.
- The string and substring must be consistent. Do not mix VARBINARY with CHAR or VARCHAR.
- POSITION is similar to [STRPOS](#) although POSITION allows finding by characters and by octet.
- If the string is not found, the return value is zero.

Examples

```
=> SELECT POSITION('é' IN 'étudiant' USING CHARACTERS);
position
-----
         1
(1 row)

=> SELECT POSITION('ß' IN 'straße' USING OCTETS);
positionb
```

```

-----
      5
(1 row)
=> SELECT POSITION('c' IN 'abcd' USING CHARACTERS);
   position
-----
      3
(1 row)
=> SELECT POSITION(VARBINARY '456' IN VARBINARY '123456789');
   position
-----
      4
(1 row)
SELECT POSITION('n' in 'León') as 'default',
       POSITIONB('León', 'n') as 'POSITIONB',
       POSITION('n' in 'León' USING CHARACTERS) as 'pos_chars',
       POSITION('n' in 'León' USING OCTETS) as 'pos_oct', INSTR('León', 'n'),
       INSTRB('León', 'n'), REGEXP_INSTR('León', 'n');
 default | POSITIONB | pos_chars | pos_oct | INSTR | INSTRB | REGEXP_INSTR
-----+-----+-----+-----+-----+-----+-----
      4 |         |      5 |      4 |      5 |      4 |           4
(1 row)

```

POSITIONB

Returns an INTEGER value representing the octet location of a specified substring with a string (counting from one).

Behavior Type

Immutable

Syntax

POSITIONB (*string*, *substring*)

Parameters

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
<i>substring</i>	(CHAR or VARCHAR) is the substring to locate

Examples

```
=> SELECT POSITIONB('straße', 'ße');
POSITIONB
-----
          5
(1 row)

=> SELECT POSITIONB('étudiant', 'é');
POSITIONB
-----
          1
(1 row)
```

QUOTE_IDENT

Returns the given string, suitably quoted, to be used as an [identifier](#) in a SQL statement string. Quotes are added only if necessary; that is, if the string contains non-identifier characters, is a SQL [keyword](#), such as '1time', 'Next week' and 'Select'. Embedded double quotes are doubled.

Behavior Type

Immutable

Syntax

```
QUOTE_IDENT( string )
```

Parameters

<i>string</i>	String to quote.
---------------	------------------

Notes

- SQL identifiers, such as table and column names, are stored as created, and references to them are resolved using case-insensitive compares. Thus, you do not need to double-quote

mixed-case identifiers.

- Vertica quotes all currently-reserved keywords, even those not currently being used.

Examples

Quoted identifiers are case-insensitive, and Vertica does not supply the quotes:

```
=> SELECT QUOTE_IDENT('VErtIcA');
QUOTE_IDENT
-----
VErtIcA
(1 row)

=> SELECT QUOTE_IDENT('Vertica database');
QUOTE_IDENT
-----
"Vertica database"
(1 row)
```

Embedded double quotes are doubled:

```
=> SELECT QUOTE_IDENT('Vertica "!" database');
QUOTE_IDENT
-----
"Vertica ""!"" database"
(1 row)
```

The following example uses the SQL keyword, SELECT; results are double quoted:

```
=> SELECT QUOTE_IDENT('select');
QUOTE_IDENT
-----
"select"
(1 row)
```

QUOTE_LITERAL

Returns the given string, suitably quoted, to be used as a string literal in a SQL statement string. Embedded single quotes and backslashes are doubled.

Behavior Type

Immutable

Syntax

QUOTE_LITERAL (*string*)

Parameters

<i>string</i>	String to convert to a string literal.
---------------	--

Notes

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, 'You''re here!'. This is the SQL standard representation and is preferred over the form, 'You\'re here!', as backslashes are not parsed as before.

Examples

```
=> SELECT QUOTE_LITERAL('You''re here!');
   QUOTE_LITERAL
-----
'You''re here!'
(1 row)
```

See Also

- [Character String Literals](#)

REPEAT

Replicates a string the specified number of times, and concatenates the replicated values as a single string. The return value can be up to 65000 bytes in length. If the length of *string* * *count* is greater than 65000 bytes, Vertica silently truncates the results.

Behavior Type

Immutable

Syntax

```
REPEAT ( 'string', count )
```

Parameters

<i>string</i>	The string to repeat, one of the following: CHAR, VARCHAR, BINARY or VARBINARY.
<i>count</i>	An integer expression that specifies how many times to repeat <i>string</i> .

Examples

The following example repeats `vmart` three times:

```
=> SELECT REPEAT ('vmart', 3);
      REPEAT
-----
vmartvmartvmart
(1 row)
```

REPLACE

Replaces all occurrences of characters in a string with another set of characters.

Behavior Type

Immutable

Syntax

```
REPLACE ('string', 'target', 'replacement' )
```

Parameters

<i>string</i>	The string to modify.
<i>target</i>	The characters in <i>string</i> to replace.
<i>replacement</i>	The characters to replace <i>target</i> .

Examples

```
=> SELECT REPLACE('Documentation%20Library', '%20', ' ');
      REPLACE
-----
Documentation Library
(1 row)

=> SELECT REPLACE('This & That', '&', 'and');
      REPLACE
-----
This and That
(1 row)

=> SELECT REPLACE('straße', 'ß', 'ss');
      REPLACE
-----
strasse
(1 row)
```

RIGHT

Returns the specified characters from the right side of a string.

Behavior Type

Immutable

Syntax

```
RIGHT ( string , length )
```

Parameters

<i>string</i>	(CHAR or VARCHAR) is the string to return.
<i>length</i>	Is an INTEGER value that specifies the count of characters to return.

Examples

The following command returns the last three characters of the string 'vertica':

```
=> SELECT RIGHT('vertica', 3);
RIGHT
-----
ica
(1 row)
```

The following command returns the last two characters of the string 'straße':

```
=> SELECT RIGHT('straße', 2);
RIGHT
-----
ße
(1 row)
```

See Also

- [SUBSTR](#)

RPAD

Returns a VARCHAR value representing a string of a specific length filled on the right with specific characters.

Behavior Type

Immutable

Syntax

`RPAD (expression , Length [, fill])`

Parameters

<i>expression</i>	(CHAR OR VARCHAR) specifies the string to fill
<i>Length</i>	(INTEGER) specifies the number of characters to return
<i>fill</i>	(CHAR OR VARCHAR) specifies the repeating string of characters with which to fill the output string. The default is the space character.

Examples

```
=> SELECT RPAD('database', 15, 'xzy');
      RPAD
-----
databasexzyxzyx
(1 row)
```

If the string is already longer than the specified length it is truncated on the right:

```
=> SELECT RPAD('database', 6, 'xzy');
      RPAD
-----
databa
(1 row)
```

RTRIM

Returns a VARCHAR value representing a string with trailing blanks removed from the right side (end).

Behavior Type

Immutable

Syntax

```
RTRIM ( expression [ , characters ] )
```

Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to trim
<i>characters</i>	(CHAR or VARCHAR) specifies the characters to remove from the right side of <i>expression</i> . The default is the space character.

Examples

```
=> SELECT RTRIM('trimzzzyyyyyxxxxxxxx', 'yz');  
RTRIM  
-----  
trim  
(1 row)
```

See Also

- [BTRIM](#)
- [LTRIM](#)
- [TRIM](#)

SHA1

Uses the US Secure Hash Algorithm 1 to calculate the SHA1 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA1 (*string*)

Parameters

<i>string</i>	The VARCHAR or VARBINARY string to be calculated.
---------------	---

Examples

The following examples calculate the SHA1 hash of the provided strings:

```
=> SELECT SHA1('123');
           SHA1
-----
40bd001563085fc35165329ea1ff5c5ecbdbbbeeef
(1 row)
```

```
=> SELECT SHA1('Vertica'::bytea);
           SHA1
-----
ee2cff8d3444995c6c301546c4fc5ee152d77c11
(1 row)
```

See Also

- [MD5](#)
- [SHA224](#)
- [SHA256](#)
- [SHA384](#)
- [SHA512](#)

SHA224

Uses the US Secure Hash Algorithm 2 to calculate the SHA224 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

```
SHA224 ( string )
```

Parameters

<i>string</i>	The VARCHAR or VARBINARY string to be calculated.
---------------	---

Examples

The following examples calculate the SHA224 hash of the provided strings:

```
=> SELECT SHA224('abc');
           SHA224
-----
78d8045d684abd2eece923758f3cd781489df3a48e1278982466017f
(1 row)
```

```
=> SELECT SHA224('Vertica'::bytea);
           SHA224
-----
135ac268f64ff3124aeceb3cc0af0a29fd600a3be8e29ed97e45e25
(1 row)
```

```
=> SELECT sha224(' '::varbinary) = 'd14a028c2a3a2bc9476102bb288234c415a2b01f828ea62ac5b3e42f' AS
"TRUE";
           TRUE
-----
t
(1 row)
```

See Also

- [MD5](#)
- [SHA1\(\)](#)
- [SHA256\(\)](#)
- [SHA384\(\)](#)
- [SHA512\(\)](#)

SHA256

Uses the US Secure Hash Algorithm 2 to calculate the SHA256 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA256 (*string*)

Parameters

<i>string</i>	The VARCHAR or VARBINARY string to be calculated.
---------------	---

Examples

The following examples calculate the SHA256 hash of the provided strings:

```
=> SELECT SHA256('abc');  
SHA256
```

```
-----  
a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3  
(1 row)
```

```
=> SELECT SHA256('Vertica'::bytea);  
          SHA256
```

```
-----  
9981b0b7df9f5be06e9e1a7f4ae2336a7868d9ab522b9a6ca6a87cd9ed95ba53  
(1 row)
```

```
=> SELECT sha256('') = 'e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855' AS "TRUE";  
TRUE
```

```
-----  
t  
(1 row)
```

See Also

- [MD5](#)
- [SHA1](#)
- [SHA224](#)
- [SHA384](#)
- [SHA512](#)

SHA384

Uses the US Secure Hash Algorithm 2 to calculate the SHA384 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

```
SHA384 ( string )
```

Parameters

<i>string</i>	The VARCHAR or VARBINARY string to be calculated.
---------------	---

Examples

The following examples calculate the SHA384 hash of the provided strings:

```
=> SELECT SHA384('123');
                                     SHA384
-----
9a0a82f0c0cf31470d7affede3406cc9aa8410671520b727044eda15b4c25532a9b5cd8aaf9cec4919d76255b6bf00f
(1 row)
```

```
=> SELECT SHA384('Vertica'::bytea);
                                     SHA384
-----
3431a717dc3289862bbd636a064d26980b47ebe4684b800cff4756f0c24985866ef97763eafd548fedb0ce28722c96bb
(1 row)
```

See Also

- [MD5](#)
- [SHA1](#)
- [SHA224](#)
- [SHA256](#)
- [SHA512](#)

SHA512

Uses the US Secure Hash Algorithm 2 to calculate the SHA512 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA512 (*string*)

Parameters

<i>string</i>	The VARCHAR or VARBINARY string to be calculated.
---------------	---

Examples

The following examples calculate the SHA512 hash of the provided strings:

```
=> SELECT SHA512('123');
                                     SHA512
-----
3c9909afec25354d551dae21590bb26e38d53f2173b8d3dc3eee4c047e7ab1c1eb8b85103e3be7ba613b31bb5c9c36214dc9f
14a42fd7a2fdb84856bca5c44c2
(1 row)
```

```
=> SELECT SHA512('Vertica'::bytea);
                                     SHA512
-----
c4ee2b2d17759226a3897c9c30d7c6df1145c4582849bb5191ee140bce05b83d3d869890cc3619b534fea6f97ff28a739d8b5
68a5ade66e756b3243ef97d3f00
(1 row)
```

See Also

- [MD5](#)
- [SHA1](#)
- [SHA224](#)
- [SHA256](#)
- [SHA384](#)

SPACE

Returns the specified number of blank spaces, typically for insertion into a character string.

Behavior Type

Immutable

Syntax

`SPACE(n)`

Parameters

<i>n</i>	An integer argument that specifies how many spaces to insert.
----------	---

Example

The following example concatenates strings *x* and *y* with 10 spaces inserted between them:

```
=> SELECT 'x' || SPACE(10) || 'y' AS Ten_spaces;
   Ten_spaces
-----
x          y
(1 row)
```

SPLIT_PART

Splits string on the delimiter and returns the location of the beginning of the given field (counting from one).

Behavior Type

Immutable

Syntax

```
SPLIT_PART ( string , delimiter , field )
```

Parameters

<i>string</i>	Is the argument string.
<i>delimiter</i>	Is the given delimiter.
<i>field</i>	(INTEGER) is the number of the part to return.

Notes

Use this with the character form of the subfield.

Examples

The specified integer of 2 returns the second string, or def.

```
=> SELECT SPLIT_PART('abc~@~def~@~ghi', '~@~', 2);
   SPLIT_PART
-----
      def
(1 row)
```

In the next example, specify 3, which returns the third string, or 789.

```
=> SELECT SPLIT_PART('123~|~456~|~789', '~|~', 3);
   SPLIT_PART
-----
      789
(1 row)
```

The tildes are for readability only. Omitting them returns the same results:

```
=> SELECT SPLIT_PART('123|456|789', '|', 3);
   SPLIT_PART
-----
      789
(1 row)
```

See what happens if you specify an integer that exceeds the number of strings: No results.

```
=> SELECT SPLIT_PART('123|456|789', '|', 4);
   SPLIT_PART
-----

(1 row)
```

The previous result is not null, it is an empty string.

```
=> SELECT SPLIT_PART('123|456|789', '|', 4) IS NULL;
   ?column?
-----
         f
(1 row)
```

If `SPLIT_PART` had returned `NULL`, `LENGTH` would have returned null.

```
=> SELECT LENGTH (SPLIT_PART('123|456|789', '|', 4));
   LENGTH
-----
         0
(1 row)
```

SPLIT_PARTB

Splits string on the delimiter and returns the location of the beginning of the given field (counting from one). The `VARCHAR` arguments are treated as octets rather than UTF-8 characters.

Behavior Type

Immutable

Syntax

`SPLIT_PARTB (string , delimiter , field)`

Parameters

<i>string</i>	(VARCHAR) Is the argument string.
<i>delimiter</i>	(VARCHAR) Is the given delimiter.
<i>field</i>	(INTEGER) is the number of the part to return.

Notes

Use this function with the character form of the subfield.

Examples

The specified integer of 3 returns the third string, or `souçon`.

```
=> SELECT SPLIT_PARTB('straße~@~café~@~souçon', '~@~', 3);
   SPLIT_PARTB
-----
   souçon
(1 row)
```

The tildes are for readability only. Omitting them returns the same results:

```
=> SELECT SPLIT_PARTB('straße @ café @ souçon', '@', 3);
   SPLIT_PARTB
-----
   souçon
(1 row)
```

See what happens if you specify an integer that exceeds the number of strings: No results.

```
=> SELECT SPLIT_PARTB('straße @ café @ souçon', '@', 4);
   SPLIT_PARTB
-----
(1 row)
```

The above result is not null, it is an empty string.

```
=> SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 4) IS NULL;  
?column?  
-----  
f  
(1 row)
```

STRPOS

Returns an INTEGER value representing the character location of a specified substring within a string (counting from one).

Behavior Type

Immutable

Syntax

```
STRPOS ( string , substring )
```

Parameters

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
<i>substring</i>	(CHAR or VARCHAR) is the substring to locate

Notes

STRPOS is similar to [POSITION](#) although POSITION allows finding by characters and by octet.

If the string is not found, the return value is zero.

Examples

```
=> SELECT STRPOS('abcd', 'c');  
STRPOS  
-----  
3  
(1 row)
```

STRPOSB

Returns an INTEGER value representing the location of a specified substring within a string, counting from one, where each octet in the string is counted (as opposed to characters).

Behavior Type

Immutable

Syntax

`STRPOSB (string , substring)`

Parameters

<i>string</i>	(CHAR or VARCHAR) is the string in which to locate the substring
<i>substring</i>	(CHAR or VARCHAR) is the substring to locate

Notes

STRPOSB is identical to [POSITIONB](#) except for the order of the arguments.

Examples

```
=> SELECT STRPOSB('straße', 'e');
   STRPOSB
-----
         7
(1 row)

=> SELECT STRPOSB('étudiant', 'tud');
   STRPOSB
-----
         3
(1 row)
```

SUBSTR

Returns VARCHAR or VARBINARY value representing a substring of a specified string.

Behavior Type

Immutable

Syntax

```
SUBSTR ( string , position [ , extent ] )
```

Parameters

<i>string</i>	(CHAR/VARCHAR or BINARY/VARBINARY) is the string from which to extract a substring.
<i>position</i>	(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one by characters).
<i>extent</i>	(INTEGER or DOUBLE PRECISION) is the length of the substring to extract (in characters). The default is the end of the string.

Notes

SUBSTR truncates DOUBLE PRECISION input values.

Examples

```
=> SELECT SUBSTR('abc'::binary(3),1);
 substr
-----
 abc
(1 row)

=> SELECT SUBSTR('123456789', 3, 2);
 substr
```

```
-----  
34  
(1 row)  
=> SELECT SUBSTR('123456789', 3);  
substr  
-----  
3456789  
(1 row)  
=> SELECT SUBSTR(TO_BITSTRING(HEX_TO_BINARY('0x10')), 2, 2);  
substr  
-----  
00  
(1 row)  
=> SELECT SUBSTR(TO_HEX(10010), 2, 2);  
substr  
-----  
71  
(1 row)
```

SUBSTRB

Returns an octet value representing the substring of a specified string.

Behavior Type

Immutable

Syntax

```
SUBSTRB ( string , position [ , extent ] )
```

Parameters

<i>string</i>	(CHAR/VARCHAR) is the string from which to extract a substring.
<i>position</i>	(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one in octets).
<i>extent</i>	(INTEGER or DOUBLE PRECISION) is the length of the substring to extract (in octets). The default is the end of the string

Notes

- This function treats the multibyte character string as a string of octets (bytes) and uses octet numbers as incoming and outgoing position specifiers and lengths. The strings themselves are type VARCHAR, but they are treated as if each octet were a separate character.
- SUBSTRB truncates DOUBLE PRECISION input values.

Examples

```
=> SELECT SUBSTRB('soupçon', 5);
SUBSTRB
-----
çon
(1 row)

=> SELECT SUBSTRB('soupçon', 5, 2);
SUBSTRB
-----
ç
(1 row)
```

Vertica returns the following error message if you use BINARY/VARBINARY:

```
=>SELECT SUBSTRB('abc'::binary(3),1);
ERROR: function substrb(binary, int) does not exist, or permission is denied for substrb(binary, int)
HINT: No function matches the given name and argument types. You may need to add explicit type casts.
```

SUBSTRING

Returns a value representing a substring of the specified string at the given position, given a value, a position, and an optional length. SUBSTRING truncates DOUBLE PRECISION input values.

Behavior Type

Immutable if USING OCTETS, stable otherwise.

Syntax

```
SUBSTRING ( string , position [ , length ]
... [USING {CHARACTERS | OCTETS } ] )
```

```
SUBSTRING ( string FROM position [ FOR length ]
... [USING { CHARACTERS | OCTETS } ] )
```

Parameters

<i>string</i>	(CHAR/VARCHAR or BINARY/VARBINARY) is the string from which to extract a substring
<i>position</i>	(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one by either characters or octets). (The default is characters.) If position is greater than the length of the given value, an empty value is returned.
<i>length</i>	(INTEGER or DOUBLE PRECISION) is the length of the substring to extract in either characters or octets. (The default is characters.) The default is the end of the string. If a length is given the result is at most that many bytes. The maximum length is the length of the given value less the given position. If no length is given or if the given length is greater than the maximum length then the length is set to the maximum length.
USING CHARACTERS OCTETS	Determines whether the value is expressed in characters (the default) or octets.

Examples

```
=> SELECT SUBSTRING('abc'::binary(3),1);
  substring
-----
  abc
(1 row)

=> SELECT SUBSTRING('soupçon', 5, 2 USING CHARACTERS);
  substring
-----
  çö
(1 row)

=> SELECT SUBSTRING('soupçon', 5, 2 USING OCTETS);
  substring
-----
  ç
(1 row)
```

If you use a negative position, then the functions starts at a non-existent position. In this example, that means counting eight characters starting at position -4. So the function starts at the empty position -4 and counts five characters, including a position for zero which is also empty. This returns three characters.

```
=> SELECT SUBSTRING('1234567890', -4, 8);
 substring
-----
123
(1 row)
```

TO_BITSTRING

Returns a VARCHAR that represents the given VARBINARY value in bitstring format. This function is the inverse of [BITSTRING_TO_BINARY](#).

Behavior Type

Immutable

Syntax

```
TO_BITSTRING ( expression )
```

Parameters

<i>expression</i>	The VARCHAR string to process.
-------------------	--------------------------------

Examples

```
=> SELECT TO_BITSTRING('ab'::BINARY(2));
 to_bitstring
-----
0110000101100010
(1 row)

=> SELECT TO_BITSTRING(HEX_TO_BINARY('0x10'));
 to_bitstring
-----
00010000
(1 row)
```

```
=> SELECT TO_BITSTRING(HEX_TO_BINARY('0xF0'));
   to_bitstring
-----
11110000
(1 row)
```

See Also

[BITCOUNT](#)

TO_HEX

Returns a VARCHAR or VARBINARY representing the hexadecimal equivalent of a number. This function is the inverse of [HEX_TO_BINARY](#).

Behavior Type

Immutable

Syntax

```
TO_HEX ( number )
```

Parameters

<i>number</i>	An INTEGER or VARBINARY value to convert to hexadecimal. If you supply a VARBINARY argument, the function's return value is not preceded by 0x.
---------------	---

Examples

```
=> SELECT TO_HEX(123456789);
   TO_HEX
-----
75bcd15
(1 row)
```

For VARBINARY inputs, the returned value is not preceded by 0x. For example:

```
=> SELECT TO_HEX('ab'::binary(2));
   TO_HEX
-----
   6162
(1 row)
```

TRANSLATE

Replaces individual characters in *string_to_replace* with other characters.

Behavior Type

Immutable

Syntax

```
TRANSLATE ( string_to_replace , from_string , to_string );
```

Parameters

<i>string_to_replace</i>	String to be translated.
<i>from_string</i>	Contains characters that should be replaced in <i>string_to_replace</i> .
<i>to_string</i>	Any character in <i>string_to_replace</i> that matches a character in <i>from_string</i> is replaced by the corresponding character in <i>to_string</i> .

Example

```
=> SELECT TRANSLATE('straße', 'ß', 'ss');
   TRANSLATE
-----
   strase
(1 row)
```

TRIM

Combines the BTRIM, LTRIM, and RTRIM functions into a single function.

Behavior Type

Immutable

Syntax

```
TRIM ( [ [ LEADING | TRAILING | BOTH ] characters FROM ] expression )
```

Parameters

LEADING	Removes the specified characters from the left side of the string
TRAILING	Removes the specified characters from the right side of the string
BOTH	Removes the specified characters from both sides of the string (default)
<i>characters</i>	(CHAR or VARCHAR) specifies the characters to remove from <i>expression</i> . The default is the space character.
<i>expression</i>	(CHAR or VARCHAR) is the string to trim

Examples

```
=> SELECT '-' || TRIM(LEADING 'x' FROM 'xxdatabasexx') || '-';
?column?
-----
-databasexx-
(1 row)

=> SELECT '-' || TRIM(TRAILING 'x' FROM 'xxdatabasexx') || '-';
?column?
-----
-xxdatabase-
(1 row)

=> SELECT '-' || TRIM(BOTH 'x' FROM 'xxdatabasexx') || '-';
?column?
-----
-database-
(1 row)

=> SELECT '-' || TRIM('x' FROM 'xxdatabasexx') || '-';
?column?
-----
```

```
-database-
(1 row)
=> SELECT '-' || TRIM(LEADING FROM ' database ' ) || '-';
?column?
-----
-database -
(1 row)
=> SELECT '-' || TRIM(' database ') || '-'; ?column?
-----
-database-
(1 row)
```

See Also

- [BTRIM](#)
- [LTRIM](#)
- [RTRIM](#)

UPPER

Returns a VARCHAR value containing the argument converted to uppercase letters.

Starting in Release 5.1, this function treats the `string` argument as a UTF-8 encoded string, rather than depending on the collation setting of the locale (for example, `collation=binary`) to identify the encoding.

Behavior Type

stable

Syntax

```
UPPER ( expression )
```

Parameters

<i>expression</i>	CHAR or VARCHAR containing the string to convert
-------------------	--

Notes

UPPER is restricted to 32500 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

Examples

```
=> SELECT UPPER('AbCdEFG');
      UPPER
-----
ABCDEF
(1 row)
=> SELECT UPPER('étudiant');
      UPPER
-----
ÉTUDIANT
(1 row)
```

UPPERB

Returns a character string with each ASCII character converted to uppercase. Multibyte characters are not converted and are skipped.

Behavior Type

Immutable

Syntax

```
UPPERB ( expression )
```

Parameters

<i>expression</i>	(CHAR or VARCHAR) is the string to convert
-------------------	--

Examples

In the following example, the multibyte UTF-8 character é is not converted to uppercase:

```
=> SELECT UPPERB('étudiant');
      UPPERB
-----
     étUDIANT
(1 row)

=> SELECT UPPERB('AbCdEfG');
      UPPERB
-----
     ABCDEFG
(1 row)

=> SELECT UPPERB('The Vertica Database');
      UPPERB
-----
     THE VERTICA DATABASE
(1 row)
```

V6_ATON

Converts an IPv6 address represented as a character string to a binary string.

Behavior Type

Immutable

Syntax

V6_ATON (*expression*)

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Notes

The following syntax converts an IPv6 address represented as the character string A to a binary string B.

V6_ATON trims any spaces from the right of A and calls the Linux function [inet_pton](#).

```
=> V6_ATON(VARCHAR A) -> VARBINARY(16) B
```

If A has no colons it is prepended with '::ffff:'. If A is NULL, too long, or if `inet_pton` returns an error, the result is NULL.

Examples

```
=> SELECT V6_ATON('2001:DB8::8:800:200C:417A');
           v6_aton
-----
\001\015\270\000\000\000\000\000\000\010\010\000 \014Az
(1 row)

=> SELECT V6_ATON('1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\377\377\001\002\003\004
(1 row)
SELECT TO_HEX(V6_ATON('2001:DB8::8:800:200C:417A'));
           to_hex
-----
20010db800000000000000800200c417a
(1 row)

=> SELECT V6_ATON('::1.2.3.4');
           v6_aton
-----
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\001\002\003\004
(1 row)
```

See Also

- [V6_NTOA](#)

V6_NTOA

Converts an IPv6 address represented as varbinary to a character string.

Behavior Type

Immutable

Syntax

V6_NTOA (*expression*)

Parameters

<i>expression</i>	(VARBINARY) is the binary string to convert.
-------------------	--

Notes

The following syntax converts an IPv6 address represented as VARBINARY B to a string A.

V6_NTOA right-pads B to 16 bytes with zeros, if necessary, and calls the Linux function [inet_ntop](#).

```
=> V6_NTOA(VARBINARY B) -> VARCHAR A
```

If B is NULL or longer than 16 bytes, the result is NULL.

Vertica automatically converts the form '::ffff:1.2.3.4' to '1.2.3.4'.

Examples

```
=> SELECT V6_NTOA(' \001\015\270\000\000\000\000\000\010\010\000 \014Az');
      v6_ntoa
-----
2001:db8::8:800:200c:417a
(1 row)

=> SELECT V6_NTOA(V6_ATON('1.2.3.4'));
      v6_ntoa
-----
1.2.3.4
(1 row)

=> SELECT V6_NTOA(V6_ATON('::1.2.3.4'));
      v6_ntoa
-----
```

```
:::1.2.3.4  
(1 row)
```

See Also

- [V6_ATON](#)

V6_SUBNETA

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a binary or alphanumeric IPv6 address.

Behavior Type

Immutable

Syntax

```
V6_SUBNETA ( expression1, expression2 )
```

Parameters

<i>expression1</i>	(VARBINARY or VARCHAR) is the string to calculate.
<i>expression2</i>	(INTEGER) is the size of the subnet.

Notes

The following syntax calculates a subnet address in CIDR format from a binary or varchar IPv6 address.

V6_SUBNETA masks a binary IPv6 address B so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared. It then converts to an alphanumeric IPv6 address, appending a slash and N.

```
=> V6_SUBNETA(BINARY B, INT8 N) -> VARCHAR C
```

The following syntax calculates a subnet address in CIDR format from an alphanumeric IPv6 address.

```
=> V6_SUBNETA(VARCHAR A, INT8 N) -> V6_SUBNETA(V6_ATON(A), N) -> VARCHAR C
```

Examples

```
=> SELECT V6_SUBNETA(V6_ATON('2001:db8::8:800:200c:417a'), 28);
   v6_subneta
-----
2001:db8::/28
(1 row)
```

See Also

- [V6_SUBNETN](#)

V6_SUBNETN

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a varbinary or alphanumeric IPv6 address.

Behavior Type

Immutable

Syntax

```
V6_SUBNETN ( expression1, expression2 )
```

Parameters

<i>expression1</i>	(VARBINARY or VARCHAR) is the string to calculate. Notes: <ul style="list-style-type: none">• V6_SUBNETN(<VARBINARY>, <INTEGER>) returns VARBINARY.
--------------------	---

	<p>OR</p> <ul style="list-style-type: none"> • <code>V6_SUBNETN(<VARCHAR>, <INTEGER>)</code> returns <code>VARBINARY</code>, after using <code>V6_ATON</code> to convert the <code><VARCHAR></code> string to <code><VARBINARY></code>.
<i>expression2</i>	<code>(INTEGER)</code> is the size of the subnet.

Notes

The following syntax masks a `BINARY` IPv6 address `B` so that the `N` left-most bits of `S` form a subnet address, while the remaining right-most bits are cleared.

`V6_SUBNETN` right-pads `B` to 16 bytes with zeros, if necessary and masks `B`, preserving its `N`-bit subnet prefix.

```
=> V6_SUBNETN(VARBINARY B, INT8 N) -> VARBINARY(16) S
```

If `B` is `NULL` or longer than 16 bytes, or if `N` is not between 0 and 128 inclusive, the result is `NULL`.

`S` = `[B]/N` in [Classless Inter-Domain Routing](#) notation (CIDR notation).

The following syntax masks an alphanumeric IPv6 address `A` so that the `N` leftmost bits form a subnet address, while the remaining rightmost bits are cleared.

```
=> V6_SUBNETN(VARCHAR A, INT8 N) -> V6_SUBNETN(V6_ATON(A), N) -> VARBINARY(16) S
```

Example

This example returns `VARBINARY`, after using `V6_ATON` to convert the `VARCHAR` string to `VARBINARY`:

```
=> SELECT V6_SUBNETN(V6_ATON('2001:db8::8:800:200c:417a'), 28);
           v6_subnetn
-----
\001\015\260\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
```

See Also

- [V6_ATON](#)
- [V6_SUBNETA](#)

V6_TYPE

Characterizes a binary or alphanumeric IPv6 address B as an integer type.

Behavior Type

Immutable

Syntax

```
V6_TYPE ( expression )
```

Parameters

<i>expression</i>	(VARBINARY or VARCHAR) is the type to convert.
-------------------	--

Notes

V6_TYPE(VARBINARY B) returns INT8 T.

```
=> V6_TYPE(VARCHAR A) -> V6_TYPE(V6_ATON(A)) -> INT8 T
```

The IPv6 types are defined in the Network Working Group's [IP Version 6 Addressing Architecture memo](#).

GLOBAL = 0	Global unicast addresses
LINKLOCAL = 1	Link-Local unicast (and Private-Use) addresses
LOOPBACK = 2	Loopback
UNSPECIFIED = 3	Unspecified
MULTICAST = 4	Multicast

IPv4-mapped and IPv4-compatible IPv6 addresses are also interpreted, as specified in [IPv4 Global Unicast Address Assignments](#).

- For IPv4, Private-Use is grouped with Link-Local.
- If B is VARBINARY, it is right-padded to 16 bytes with zeros, if necessary.
- If B is NULL or longer than 16 bytes, the result is NULL.

Details

IPv4 (either kind):

0.0.0.0/8	UNSPECIFIED	10.0.0.0/8	LINKLOCAL
127.0.0.0/8	LOOPBACK		
169.254.0.0/16	LINKLOCAL		
172.16.0.0/12	LINKLOCAL		
192.168.0.0/16	LINKLOCAL		
224.0.0.0/4	MULTICAST		
others	GLOBAL		

IPv6:

::0/128	UNSPECIFIED	::1/128	LOOPBACK
fe80::/10	LINKLOCAL		
ff00::/8	MULTICAST		
others	GLOBAL		

Examples

```
=> SELECT V6_TYPE(V6_ATON('192.168.2.10'));
v6_type
-----
      1
(1 row)

=> SELECT V6_TYPE(V6_ATON('2001:db8::8:800:200c:417a'));
v6_type
-----
      0
(1 row)
```

See Also

- [INET_ATON](#)
- [IP Version 6 Addressing Architecture](#)
- [IPv4 Global Unicast Address Assignments](#)

System Information Functions

These functions provide system information regarding user sessions. A superuser has unrestricted access to all system information, but users can view only information about their own, current sessions.

CURRENT_DATABASE

Returns a VARCHAR value containing the name of the database to which you are connected.

Behavior Type

Immutable

Syntax

```
CURRENT_DATABASE()
```

Notes

- The parentheses following the CURRENT_DATABASE function are optional.
- This function is equivalent to DBNAME.

Examples

```
SELECT CURRENT_DATABASE();  
CURRENT_DATABASE  
-----  
VMart  
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT CURRENT_DATABASE;  
CURRENT_DATABASE  
-----
```

```
VMart  
(1 row)
```

CURRENT_SCHEMA

Returns the name of the current schema.

Behavior Type

Stable

Syntax

```
CURRENT_SCHEMA()
```

Note: You can call this function without parentheses.

Privileges

None

Examples

The following command returns the name of the current schema:

```
=> SELECT CURRENT_SCHEMA();  
current_schema  
-----  
public  
(1 row)
```

The following command returns the same results without the parentheses:

```
=> SELECT CURRENT_SCHEMA;  
current_schema  
-----  
public  
(1 row)
```

The following command shows the current schema, listed after the [current user](#), in the search path:

```
=> SHOW SEARCH_PATH;
   name      |          setting
-----+-----
 search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

See Also

- [SET SEARCH_PATH](#)

CURRENT_USER

Returns a VARCHAR containing the name of the user who initiated the current database connection.

Behavior Type

Stable

Syntax

```
CURRENT_USER()
```

Notes

- The CURRENT_USER function does not require parentheses.
- This function is useful for permission checking.
- CURRENT_USER is equivalent to [SESSION_USER](#), [USER](#), and [USERNAME](#).

Examples

```
SELECT CURRENT_USER();
CURRENT_USER
-----
dbadmin
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT CURRENT_USER;  
CURRENT_USER  
-----  
dbadmin  
(1 row)
```

DBNAME (function)

Returns a VARCHAR value containing the name of the database to which you are connected. DBNAME is equivalent to [CURRENT_DATABASE](#).

Behavior Type

Immutable

Syntax

```
DBNAME()
```

Examples

```
SELECT DBNAME();  
dbname  
-----  
VMart  
(1 row)
```

HAS_TABLE_PRIVILEGE

Indicates whether a user can access a table in a particular way. The function returns a true (t) or false (f) value.

Behavior Type

Stable

Syntax

```
HAS_TABLE_PRIVILEGE ( [ user, ] [schema-name.]table , privilege )
```

Parameters

<i>user</i>	Specifies the name or OID of a database user. The default is the CURRENT_USER .
<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>table</i>	Specifies the name or OID of a table in the logical schema. If necessary, specify the database and schema, as noted above.
<i>privilege</i>	<ul style="list-style-type: none">• SELECT Allows the user to SELECT from any column of the specified table.• INSERT Allows the user to INSERT records into the specified table and to use the COPY command to load the table.• UPDATE Allows the user to UPDATE records in the specified table.• DELETE Allows the user to delete a row from the specified table.• REFERENCES Allows the user to create a foreign key constraint (privileges required on both the referencing and referenced tables).

Privileges

A superuser can check all other user's table privileges.

Users without superuser privileges can use `HAS_TABLE_PRIVILEGE` to check:

- Any tables they own.
- Tables in a schema to which they have been granted **USAGE** privileges, and at least one other table privilege, as described in [GRANT \(Table\)](#).

Examples

```
SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'SELECT');
HAS_TABLE_PRIVILEGE
-----
t
(1 row)

SELECT HAS_TABLE_PRIVILEGE('release', 'store.store_dimension', 'INSERT');
HAS_TABLE_PRIVILEGE
-----
t
(1 row)

SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'UPDATE');
HAS_TABLE_PRIVILEGE
-----
t
(1 row)

SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'REFERENCES');
HAS_TABLE_PRIVILEGE
-----
t
(1 row)

SELECT HAS_TABLE_PRIVILEGE(45035996273711159, 45035996273711160, 'select');

HAS_TABLE_PRIVILEGE
-----
t
(1 row)
```

LIST_ENABLED_CIPHERS

Returns a list of ciphers enabled on your server.

Syntax

```
list_enabled_ciphers()
```

Example

```
=> list_enabled_ciphers()
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
```

SESSION_USER

Returns a VARCHAR containing the name of the user who initiated the current database session.

Behavior Type

Stable

Syntax

```
SESSION_USER()
```

Notes

- The SESSION_USER function does not require parentheses.
- SESSION_USER is equivalent to [CURRENT_USER](#), [USER](#), and [USERNAME](#).

Examples

```
SELECT SESSION_USER();
 session_user
-----
 dbadmin
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT SESSION_USER;
 session_user
-----
 dbadmin
(1 row)
```

USER

Returns a VARCHAR containing the name of the user who initiated the current database connection.

Behavior Type

Stable

Syntax

```
USER()
```

Notes

- The USER function does not require parentheses.
- USER is equivalent to [CURRENT_USER](#), [SESSION_USER](#), and [USERNAME](#).

Examples

```
=> SELECT USER();
current_user
-----
dbadmin
(1 row)
```

The following command returns the same results without the parentheses:

```
=> SELECT USER;
current_user
-----
dbadmin
(1 row)
```

USERNAME

Returns a VARCHAR containing the name of the user who initiated the current database connection.

Behavior Type

Stable

Syntax

```
USERNAME()
```

Notes

- This function is useful for permission checking.
- USERNAME is equivalent to [CURRENT_USER](#), [SESSION_USER](#) and [USER](#).

Examples

```
=> SELECT USERNAME();
  username
-----
  dbadmin
(1 row)
```

VERSION

Returns a VARCHAR containing an Vertica node's version information.

Behavior Type

Stable

Syntax

VERSION()

Examples

```
SELECT VERSION();
          VERSION
-----
Vertica Analytic Database v4.0.12-20100513010203
(1 row)
```

The parentheses are required. If you omit them, the system returns an error:

```
SELECT VERSION;
ERROR: column "version" does not exist
```

Timeseries Functions

Timeseries aggregate functions evaluate the values of a given set of variables over time and group those values into a window for analysis and aggregation.

One output row is produced per time slice—or per partition per time slice—if partition expressions are present.

TS_FIRST_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, `TS_FIRST_VALUE` returns the value at the start of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of `const` (linear). There is one value per time slice per partition.

Behavior Type

Immutable

Syntax

```
TS_FIRST_VALUE ( expression [ IGNORE NULLS ]  
... [, { 'CONST' | 'LINEAR' } ] )
```

Parameters

<i>expression</i>	Argument expression on which to aggregate and interpolate. <i>expression</i> is data type INTEGER or FLOAT.
IGNORE NULLS	The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in Analyzing Data for details.
'CONST' 'LINEAR'	(Default CONST) Optionally specifies the interpolation value as either constant or linear.

- **CONST**—New value are interpolated based on previous input records.
- **LINEAR**—Values are interpolated in a linear slope based on the specified time slice.

Notes

- The function returns one output row per time slice or one output row per partition per time slice if partition expressions are specified.
- Multiple time series aggregate functions can exists in the same query. They share the same gap-filling policy as defined by the [TIMESERIES Clause](#); however, each time series aggregate function can specify its own interpolation policy. For example:

```
=> SELECT slice_time, symbol,  
       TS_FIRST_VALUE(tid, 'const') fv_c,  
       TS_FIRST_VALUE(tid, 'linear') fv_l,  
       TS_LAST_VALUE(tid, 'const') lv_c  
FROM TickStore  
TIMESERIES slice_time AS '3 seconds'  
OVER(PARTITION BY symbol ORDER BY ts);
```

- You must use an **ORDER BY** clause with a **TIMESTAMP** column.

Example

For detailed examples, see [Gap Filling and Interpolation](#) in Analyzing Data.

See Also

- [TIMESERIES Clause](#)
- [TS_LAST_VALUE](#)
- [Time Series Analytics](#)

TS_LAST_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, `TS_LAST_VALUE` returns the value at the end of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of `const` (linear). There is one value per time slice per partition.

Behavior Type

Immutable

Syntax

```
TS_LAST_VALUE ( expression [ IGNORE NULLS ]  
... [, { 'CONST' | 'LINEAR' } ] )
```

Parameters

<i>expression</i>	Argument expression on which to aggregate and interpolate. <i>expression</i> is data type INTEGER or FLOAT.
IGNORE NULLS	The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in Analyzing Data for details.
'CONST' 'LINEAR'	(Default CONST) Optionally specifies the interpolation value as either constant or linear. <ul style="list-style-type: none">• CONST—New value are interpolated based on previous input records.• LINEAR—Values are interpolated in a linear slope based on the specified time slice.

Notes

- The function returns one output row per time slice or one output row per partition per time slice if partition expressions are specified.
- Multiple time series aggregate functions can exist in the same query. They share the same gap-filling policy as defined by the [TIMESERIES Clause](#); however, each time series aggregate function can specify its own interpolation policy. For example:

```
=> SELECT slice_time, symbol,  
       TS_FIRST_VALUE(bid, 'const') fv_c,  
       TS_FIRST_VALUE(bid, 'linear') fv_l,  
       TS_LAST_VALUE(bid, 'const') lv_c  
FROM TickStore  
TIMESERIES slice_time AS 3 seconds  
OVER(PARTITION BY symbol ORDER BY ts);
```

- You must use the `ORDER BY` clause with a `TIMESTAMP` column.

Example

For detailed examples, see [Gap Filling and Interpolation](#) in Analyzing Data.

See Also

- [TIMESERIES Clause](#)
- [TS_FIRST_VALUE](#)
- [Time Series Analytics](#)

URI Encode/Decode Functions

The functions in this section follow the RFC 3986 standard for percent-encoding a Universal Resource Identifier (URI).

URI_PERCENT_DECODE

Decodes a percent-encoded Universal Resource Identifier (URI) according to the RFC 3986 standard.

Syntax

```
URI_PERCENT_DECODE (expression)
```

Behavior Type

Immutable

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Examples

The following example invokes `uri_percent_decode` on the `Websites` column of the `URI` table and returns a decoded URI:

```
=> SELECT URI_PERCENT_DECODE(Websites) from URI;
      URI_PERCENT_DECODE
-----
http://www.faqs.org/rfcs/rfc3986.html x xj%a%
(1 row)
```

The following example returns the original URI in the `Websites` column and its decoded version:

```
=> SELECT Websites, URI_PERCENT_DECODE (Websites) from URI;

-----+-----
      Websites                |                URI_PERCENT_DECODE
-----+-----
http://www.faqs.org/rfcs/rfc3986.html+x%20x%6a%a%a% | http://www.faqs.org/rfcs/rfc3986.html x xj%a%

(1 row)
```

URI_PERCENT_ENCODE

Encodes a Universal Resource Identifier (URI) according to the RFC 3986 standard for percent encoding. In addition, for compatibility with older encoders this function converts '+' to space; space is converted to %20 by `uri_percent_encode`.

Syntax

`URI_PERCENT_ENCODE (expression)`

Behavior Type

Immutable

Parameters

<i>expression</i>	(VARCHAR) is the string to convert.
-------------------	-------------------------------------

Examples

The following example shows how the `uri_percent_encode` function is invoked on a the `Websites` column of the `URI` table and returns an encoded URI:

```
=> SELECT URI_PERCENT_ENCODE(Websites) from URI;

-----+-----
      URI_PERCENT_ENCODE
-----+-----
http%3A%2F%2Fexample.com%2F%3F%3D11%2F15

(1 row)
```

The following example returns the original URI in the `Websites` column and it's encoded form:

```
=> SELECT Websites, URI_PERCENT_ENCODE(Websites) from URI;      Websites      |
URI_PERCENT_ENCODE
-----+-----
http://example.com/?=11/15 | http%3A%2F%2Fexample.com%2F%3F%3D11%2F15
(1 row)
```

Vertica Meta-Functions

Vertica built-in (meta) functions access the internal state of Vertica and are used in SELECT queries with the function name and an argument (where required). These functions are not part of the SQL standard and take the following form:

```
SELECT <meta-function-name>(<args>);
```

Note: The query cannot contain other clauses, such as FROM or WHERE.

The behavior type of Vertica meta-functions is immutable.

Alphabetical List of Vertica Meta-Functions

The following list shows all of the Vertica meta-functions in alphabetical order.

Jump to letter: [A](#) - [B](#) - [C](#) - [D](#) - [E](#) - [F](#) - [G](#) - [H](#) - [I](#) - [K](#) - [L](#) - [M](#) - [N](#) - [P](#) - [R](#) - [S](#) - [V](#)

[ADVANCE_EPOCH](#)

Manually closes the current epoch and begins a new epoch.

[ALTER_LOCATION_LABEL](#)

Alters the location label.

[ALTER_LOCATION_USE](#)

Alters the type of files that can be stored at the specified storage location.

[ANALYZE_CONSTRAINTS](#)

Analyzes and reports on constraint violations within the specified scope.

[ANALYZE_CORRELATIONS](#)

Analyzes the specified tables for pairs of columns that are strongly correlated.

[ANALYZE_EXTERNAL_ROW_COUNT](#)

Calculates the exact number of rows in an external table.

[ANALYZE_STATISTICS](#)

Collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table.

[ANALYZE_WORKLOAD](#)

Runs the Workload Analyzer (WLA), a utility that analyzes system information held in system tables .

AUDIT

Estimates the raw data size of a database, schema, or table as it is counted in an audit of the database size.

AUDIT_FLEX

Estimates the ROS size of one or more flexible tables contained in a database, schema, or projection.

AUDIT_LICENSE_SIZE

Triggers an immediate audit of the database size to determine if it is in compliance with the raw data storage allowance included in your licenses.

AUDIT_LICENSE_TERM

Triggers an immediate audit to determine if the license has expired.

AWS_GET_CONFIG

Returns the current Amazon Web Services (AWS) credentials set by `AWS_SET_CONFIG` or `ALTER SESSION`.

AWS_SET_CONFIG

Gets the values from a table with your Amazon Web Services (AWS) credentials and passes them to session parameters.

BUILD_FLEXTABLE_VIEW

Creates, or re-creates, a view for a default or user-defined `_keys` table, ignoring any empty keys.

CANCEL_REBALANCE_CLUSTER

Stops any rebalance task that is currently in progress or is waiting to execute.

CANCEL_REFRESH

Cancels refresh-related internal operations initiated by `START_REFRESH` and `REFRESH`.

CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY

Changes the run-time priority of an active query.

CHANGE_RUNTIME_PRIORITY

Changes the run-time priority of a query that is actively running.

CLEAR_CACHES

Clears the internal cache files.

CLEAR_DATA_COLLECTOR

Clears all memory and disk records on the Data Collector tables and functions and resets collection statistics in the system table `DATA_COLLECTOR`.

CLEAR_HDFS_CACHES

Clears the configuration information copied from HDFS and any cached connections.

CLEAR_OBJECT_STORAGE_POLICY

Removes an existing storage policy.

[CLEAR_PROFILING](#)

Clears from memory data for the specified profiling type.

[CLEAR_PROJECTION_REFRESHES](#)

Triggers to clear information about refresh operations for projections immediately.

[CLEAR_RESOURCE_REJECTIONS](#)

Clears the content of the RESOURCE_REJECTIONS and DISK_RESOURCE_REJECTIONS system tables.

[CLOSE_ALL_RESULTSETS](#)

Closes all result set sessions within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

[CLOSE_ALL_SESSIONS](#)

Closes all external sessions except the one that issues this function.

[CLOSE_RESULTSET](#)

Closes a specific result set within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

[CLOSE_SESSION](#)

Interrupts the specified external session, rolls back the current transaction if any, and closes the socket.

[CLOSE_USER_SESSIONS](#)

Stops the session for a user, rolls back any transaction currently running, and closes the connection.

[COMPACT_STORAGE](#)

Bundles existing data (.fdb) and index (.pidx) files into the .gt file format.

[COMPUTE_FLEXTABLE_KEYS](#)

Computes the virtual columns (keys and values) from the flex table VMap data.

[COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)

Combines the functionality of BUILD_FLEXTABLE_VIEW and COMPUTE_FLEXTABLE_KEYS to compute virtual columns (keys) from the VMap data of a flex table and construct a view.

[COPY_PARTITIONS_TO_TABLE](#)

Copies partitions from one table to another.

[COPY_TABLE](#)

Copies one table to another.

[CURRENT_SCHEMA](#)

Returns the name of the current schema.

[DATA_COLLECTOR_HELP](#)

Returns online usage instructions about the Data Collector, the V_MONITOR.DATA_COLLECTOR system table, and the Data Collector control functions.

DELETE_COMMUNITY_KEY

Removes the security key from the database DFS.

DELETE_TOKENIZER_CONFIG_FILE

Deletes a tokenizer configuration file.

DESCRIBE_COMMUNITY_KEY

Retrieves the value of the security key so you can use it in a query.

DESIGNER_ADD_DESIGN_QUERIES

Reads and evaluates queries from an input file, and adds the queries that it accepts to the specified design.

DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS

Executes the specified query and evaluates results in the following columns:

DESIGNER_ADD_DESIGN_QUERY

Reads and parses the specified query, and if accepted, adds it to the design.

DESIGNER_ADD_DESIGN_TABLES

Adds the specified tables to a design.

DESIGNER_CANCEL_POPULATE_DESIGN

Cancels population or deployment operation for the specified design if it is currently running.

DESIGNER_CREATE_DESIGN

Creates a design with the specified name.

DESIGNER_DESIGN_PROJECTION_ENCODINGS

Analyzes encoding in the specified projections, creates a script to implement encoding recommendations, and optionally deploys the recommendations.

DESIGNER_DROP_ALL_DESIGNS

Removes all -related schemas associated with the current user.

DESIGNER_DROP_DESIGN

Removes the schema associated with the specified design and all its contents.

DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS

Displays the DDL statements that define the design projections to standard output.

DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT

Displays the deployment script for the specified design to standard output.

DESIGNER_RESET_DESIGN

Discards all run-specific information of the previous build or deployment of the specified design but keeps its configuration.

DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY

Populates the design and creates the design and deployment scripts.

DESIGNER_SET_ANALYZE_CORRELATIONS_MODE

Specifies how handles column correlations in a design.

DESIGNER_SET_DESIGN_KSAFETY

Sets K-safety for a comprehensive design and stores the K-safety value in the DESIGNS table.

DESIGNER_SET_DESIGN_TYPE

Specifies whether should create a comprehensive or incremental design.

DESIGNER_SET_OPTIMIZATION_OBJECTIVE

Valid only for comprehensive database designs, specifies the optimization objective uses.

DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS

Specifies whether a design can include unsegmented projections .

DESIGNER_WAIT_FOR_DESIGN

Waits for completion of operations that are populating and deploying the design.

DISABLE_DUPLICATE_KEY_ERROR

Disables error messaging when finds duplicate primary or unique key values at run time (for use with key constraints that are not automatically enabled).

DISABLE_ELASTIC_CLUSTER

Disables elastic cluster scaling, which prevents from bundling data into chunks that are easily transportable to other nodes when performing cluster resizing.

DISABLE_LOCAL_SEGMENTS

Disables local data segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes.

DISABLE_PROFILING

Disables profiling for the specified profiling type.

DISPLAY_LICENSE

Returns the terms of your license.

DO_TM_TASK

Runs a Tuple Mover operation on the specified table or projection and commits any current transaction.

DROP_EXTERNAL_ROW_COUNT

Removes external table row count statistics compiled by ANALYZE_EXTERNAL_ROW_COUNT .

DROP_LICENSE

Drops a license key from the global catalog.

DROP_LOCATION

Removes the specified storage location.

DROP_PARTITION

Drops the specified partition.

DROP_STATISTICS

Removes statistical data on database projections previously generated by ANALYZE_STATISTICS .

DUMP_CATALOG

Returns an internal representation of the catalog.

DUMP_LOCKTABLE

Returns information about deadlocked clients and the resources they are waiting for.

DUMP_PARTITION_KEYS

Dumps the partition keys of all projections in the system.

DUMP_PROJECTION_PARTITION_KEYS

Dumps the partition keys of the specified projection.

DUMP_TABLE_PARTITION_KEYS

Dumps the partition keys of all projections for the specified table.

EMPTYMAP

Constructs a new VMap with one row but without keys or data.

ENABLE_ELASTIC_CLUSTER

Enables elastic cluster scaling, which makes enlarging or reducing the size of your database cluster more efficient by segmenting a node's data into chunks that can be easily moved to other hosts.

ENABLE_LOCAL_SEGMENTS

Enables local storage segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes.

ENABLE_PROFILING

Enables profiling for the specified profiling type.

ENFORCE_OBJECT_STORAGE_POLICY

Applies object storage policies immediately, instead of waiting for the Tuple Mover to perform the next moveout.

EVALUATE_DELETE_PERFORMANCE

Evaluates projections for potential DELETE performance issues.

EXPORT_CATALOG

Generates a SQL script for recreating a physical schema design on another cluster.

EXPORT_DIRECTED_QUERIES

Generates SQL for creating directed queries from a set of input queries, and writes the SQL to the specified file or to standard output.

EXPORT_OBJECTS

Generates a SQL script you can use to recreate non-virtual catalog objects on another cluster.

EXPORT_STATISTICS

Generates database statistics in XML format from data previously collected by ANALYZE_STATISTICS .

EXPORT_TABLES

Generates a SQL script that can be used to recreate a logical schema—schemas, tables, constraints, and views—on another cluster.

FLUSH_DATA_COLLECTOR

Waits until memory logs are moved to disk and then flushes the Data Collector, synchronizing the log with the disk storage.

GET_AHM_EPOCH

Returns the number of the epoch in which the Ancient History Mark is located.

GET_AHM_TIME

Returns a TIMESTAMP value representing the Ancient History Mark .

GET_AUDIT_TIME

Reports the time when the automatic audit of database size occurs.

GET_CLIENT_LABEL

Returns the client connection label for the current session.

GET_COMPLIANCE_STATUS

Displays whether your database is in compliance with your license agreement.

GET_CURRENT_EPOCH

The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written.

GET_DATA_COLLECTOR_POLICY

Retrieves a brief statement about the retention policy for the specified component.

GET_LAST_GOOD_EPOCH

Returns the last good epoch number.

GET_NUM_ACCEPTED_ROWS

Returns the number of rows loaded into the database for the last completed load for the current session.

GET_NUM_REJECTED_ROWS

Returns the number of rows that were rejected during the last completed load for the current session.

GET_PROJECTIONS

Returns information about projections that are anchored to the specified table, as follows:.

GET_PROJECTION_STATUS

Returns information relevant to the status of a projection :.

GET_TOKENIZER_PARAMETER

Returns the configuration parameter for a given tokenizer.

HAS_ROLE

Indicates, with a Boolean value, whether a role has been assigned to a user.

IDOL_CHECK_ACL

Checks the access control list to verify that the user has permissions to access the data in the IDOL flex table and any views created from the table.

IMPORT_DIRECTED_QUERIES

Imports to the database catalog directed queries from a SQL file that was generated by EXPORT_DIRECTED_QUERIES .

IMPORT_STATISTICS

Imports statistics from the XML file that was generated by EXPORT_STATISTICS .

INSTALL_COMMUNITY_KEY

Stores the Connector Framework Service security key in the Distributed File System (DFS).

INSTALL_LICENSE

Installs the license key in the global catalog.

INTERRUPT_STATEMENT

Interrupts the specified statement in a user session, rolls back the current transaction, and writes a success or failure message to the log file.

KERBEROS_CONFIG_CHECK

Tests the Kerberos configuration of a cluster.

KERBEROS_HDFS_CONFIG_CHECK

Tests the Kerberos configuration of a cluster that uses HDFS.

LAST_INSERT_ID

Returns the last value of a column whose value is automatically incremented through AUTO_INCREMENT or IDENTITY column constraints .

MAKE_AHM_NOW

Sets the Ancient History Mark (AHM) to the greatest allowable value.

MAPAGGREGATE

Returns a LONG VARBINARY VMap with keys and value pairs supplied from two VARCHAR input columns of an existing columnar table.

MAPCONTAINSKEY

Determines whether a VMap contains a virtual column (key).

MAPCONTAINSVALUE

Determines whether a VMap contains a specific value.

MAPDELIMITEDEXTRACTOR

Extracts data with a delimiter character, and other optional arguments, returning a single VMap value.

MAPITEMS

Returns information about items in a VMap.

MAPJSONEXTRACTOR

Extracts content of repeated JSON data objects, including nested maps, or data with an outer list of JSON elements.

MAPKEYS

Returns the virtual columns (and values) present in any VMap data.

MAPKEYSINFO

Returns virtual column information from a given map.

MAPLOOKUP

Returns single-key values from VMAP data.

MAPPUT

Accepts a VMap and one or more key/value pairs and returns a new VMap with the key/value pairs added.

MAPREGEXEXTRACTOR

Extracts data from a regular expression and returns the results as a VMap.

MAPSIZE

Returns the number of virtual columns present in any VMap data.

MAPTOSTRING

Recursively builds a string representation VMap data, including nested JSON maps.

MAPVALUES

Returns a string representation of the top-level values from a VMap.

MAPVERSION

Returns the version or invalidity of any map data.

MARK_DESIGN_KSAFE

Enables or disables high availability in your environment, in case of a failure.

MATERIALIZE_FLEXTABLE_COLUMNS

Materializes virtual columns listed as key_names in the flextable_keys table you compute using either COMPUTE_FLEXTABLE_KEYS or COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW

.

MEASURE_LOCATION_PERFORMANCE

Measures disk performance for the location specified.

MOVE_PARTITIONS_TO_TABLE

Moves partitions from one table to another.

MOVE_RETIRED_LOCATION_DATA

Moves all data from either a single retired storage location or all retired storage locations in the database.

MOVE_STATEMENT_TO_RESOURCE_POOL

Attempts to move the specified query to the specified target pool.

NOTIFY

Specifies the text message to include with a notification.

PARTITION_PROJECTION

Splits ROS containers for a specified projection.

PARTITION_TABLE

Forces the database to split ROS containers in the specified table that contain multiple distinct values of the partitioning expression.

PURGE

Permanently removes delete vectors from ROS storage containers so disk space can be reused.

PURGE_PARTITION

Purges a table partition of deleted rows.

PURGE_PROJECTION

Permanently removes deleted data from physical storage so disk space can be reused.

PURGE_TABLE

Permanently removes deleted data from physical storage so disk space can be reused.

READ_CONFIG_FILE

Reads and returns the key-value pairs of all the parameters of a given tokenizer.

REALIGN_CONTROL_NODES

Chooses control nodes (spread hosts) from all cluster nodes and assigns the rest of the nodes in the cluster to a control node.

REBALANCE_CLUSTER

Rebalances the database cluster synchronously as a session foreground task.

REBALANCE_TABLE

Synchronously rebalances data in the specified table.

REENABLE_DUPLICATE_KEY_ERROR

Restores the default behavior of error reporting by reversing the effects of `DISABLE_DUPLICATE_KEY_ERROR`.

REFRESH

Performs a synchronous, optionally-targeted refresh of a specified table's projections.

REFRESH_COLUMNS

Refreshes table columns that are defined with the constraint `SET USING`.

RELEASE_ALL_JVM_MEMORY

Forces all sessions to release the memory consumed by their Java Virtual Machines (JVM).

RELEASE_JVM_MEMORY

Terminates a Java Virtual Machine (JVM), making available the memory the JVM was using.

RELOAD_SPREAD

Updates cluster changes to the catalog's spread configuration file.

RESERVE_SESSION_RESOURCE

Reserves memory resources from the general resource pool for the exclusive use of the backup and restore process.

RESET_LOAD_BALANCE_POLICY

Resets the counter each host in the cluster maintains, to track which host it will refer a client to when the native connection load balancing scheme is set to ROUNDROBIN .

RESET_SESSION

Applies your default connection string configuration settings to your current session.

RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

Restores the `_keys` table and the `_view` .

RESTORE_LOCATION

Restores a storage location that was previously retired with `RETIRE_LOCATION` .

RETIRE_LOCATION

Makes the specified storage location inactive.

S3

Identifies the source location of files in an Amazon S3 bucket.

S3EXPORT

Exports data to an Amazon S3 bucket from your cluster.

SET_AHM_EPOCH

Sets the Ancient History Mark (AHM) to the specified epoch.

SET_AHM_TIME

Sets the Ancient History Mark (AHM) to the epoch corresponding to the specified time on the initiator node.

SET_AUDIT_TIME

Sets the time that performs automatic database size audit to determine if the size of the database is compliant with the raw data allowance in your license.

SET_CLIENT_LABEL

Assigns a label to a client connection for the current session.

SET_CONFIG_PARAMETER

Specifies the value of a configuration parameter at the database level, or for a specific node.

SET_CONTROL_SET_SIZE

Specifies the number of cluster nodes on which to deploy control messaging (spread).

SET_DATA_COLLECTOR_POLICY

Sets a size restraint (memory and disk space in kilobytes) for the specified Data Collector table on all nodes.

SET_DATA_COLLECTOR_TIME_POLICY

Sets a time capacity for individual Data Collector tables on all nodes.

SET_LOAD_BALANCE_POLICY

Sets how native connection load balancing chooses a host to handle a client connection.

SET_LOCATION_PERFORMANCE

Sets disk performance for the location specified.

SET_OBJECT_STORAGE_POLICY

Creates or changes an object storage policy by associating a database object with a labeled storage location.

SET_SCALING_FACTOR

Sets the scaling factor that determines the number of storage containers used when rebalancing the database and when using local data segmentation is enabled.

SET_TOKENIZER_PARAMETER

Configures the tokenizer parameters.

SHOW_PROFILING_CONFIG

Shows whether profiling is enabled.

SHUTDOWN

Forces a database to shut down, even if there are users connected.

SLEEP

Waits a specified number of seconds before executing another statement or command.

START_REBALANCE_CLUSTER

Asynchronously rebalances the database cluster as a background task.

START_REFRESH

For the current schema, transfers data to projections that are not able to participate in query execution due to missing or out-of-date data.

SWAP_PARTITIONS_BETWEEN_TABLES

Swaps partitions between two tables.

SYNC_WITH_HCATALOG_SCHEMA

Copies the structure of a Hive database schema available through the HCatalog Connector to a schema.

SYNC_WITH_HCATALOG_SCHEMA_TABLE

Copies the structure of a single table in a Hive database schema available through the HCatalog Connector to a table.

VALIDATE_STATISTICS

Validates statistics in the XML file generated by the EXPORT_STATISTICS command.

[VERIFY_HADOOP_CONF_DIR](#)

Verifies that the Hadoop configuration that is used to access HDFS is valid on all nodes.

AWS Library Functions

This section contains the functions associated with the Vertica library for Amazon Web Services (AWS).

AWS_GET_CONFIG

Returns the current Amazon Web Services (AWS) credentials set by [AWS_SET_CONFIG](#) or ALTER SESSION.

Syntax

```
AWS_GET_CONFIG( 'parameter' )
```

parameter

aws_id
aws_secret
aws_region
aws_ca_path
aws_ca_bundle
aws_proxy
aws_verbose
aws_max_send_speed
aws_max_recv_speed

Parameters

aws_id	Retrieves the value for the 20-character AWS access key used to authenticate your account
aws_secret	Retrieves the value for the 40-character AWS secret access key used to authenticate your account
aws_region	Retrieves the region where your AWS bucket is located. See the AWS Documentation for the full list of values. Default value: us-east-1
aws_ca_path	Retrieves the path Vertica uses to look up SSL server certificates..

<code>aws_ca_bundle</code>	Retrieves the path Vertica uses to look up an SSL server certificate bundle.
<code>aws_proxy</code>	A string value that lets you set an HTTP/HTTPS proxy for the AWS library.
<code>aws_verbose</code>	When enabled, logs libcurl debug messages to dbLog.
<code>aws_max_send_speed</code>	Retrieves the value for the maximum transfer speed when sending data to AWS S3, in bytes per second.
<code>aws_max_recv_speed</code>	Retrieves the value for the maximum transfer speed for receiving data to AWS S3, in bytes per second.

Examples

This example retrieves a stored AWS access key in a session.

```
=> SELECT AWS_GET_CONFIG('aws_id');
   aws_get_config
-----
AKABCOEXAMPLEPKPXYZQ
(1 row)
```

See Also

- [AWS Library Parameters](#)
- [Vertica AWS Library](#)

AWS_SET_CONFIG

Gets the values from a table with your Amazon Web Services (AWS) credentials and passes them to session parameters.

Syntax

```
AWS_SET_CONFIG( 'parameter' = 'value')
```

parameter

```
aws_id
aws_secret
```

aws_region
aws_ca_path
aws_ca_bundle
aws_proxy
aws_verbose
aws_max_send_speed
aws_max_recv_speed

Parameters

aws_id	Specifies the 20-character AWS access key used to authenticate your account.
aws_secret	Specifies the 40-character AWS secret access key used to authenticate your account.
aws_region	<p>Specifies the region where your AWS bucket is located. See the AWS Documentation for the full list of values.</p> <p>You can configure <code>aws_region</code> with only one region. To access buckets in multiple regions, reset the parameter each time you change regions.</p> <p>Default value: <code>us-east-1</code></p>
aws_ca_path	<p>The path Vertica uses to look up SSL server certificates.</p> <p>Default value: System-dependent</p>
aws_ca_bundle	<p>The path Vertica uses to look up an SSL server certificate bundle.</p> <p>Default value: System-dependent</p>
aws_proxy	A string value that lets you set an HTTP/HTTPS proxy for the AWS library.
aws_verbose	<p>When enabled, logs libcurl debug messages to dbLog.</p> <p>Default value: <code>false</code></p>
aws_max_send_speed	<p>Sets the maximum transfer speed for sending data to AWS S3, in bytes per second.</p> <p>Default value: unlimited</p>
aws_max_recv_speed	<p>Sets the maximum transfer speed when receiving data to AWS S3, in bytes per second.</p> <p>Default value: unlimited</p>

Examples

Configure session parameters for an AWS access key and secret access key with credentials in table `keychain`:

```
=> SELECT AWS_SET_CONFIG('aws_id', accesskey), AWS_SET_CONFIG('aws_secret', secretaccesskey) FROM
keychain;
aws_set_config | aws_set_config
-----+-----
aws_id        | aws_secret
(1 row)
```

See Also

- [AWS_GET_CONFIG](#)
- [AWS Library Parameters](#)
- [Vertica AWS Library](#)

S3

Identifies the source location of files in an Amazon S3 bucket. Use the `S3` function in conjunction with a `COPY` statement to import data into a Vertica cluster from an S3 object.

Syntax

```
S3( url='object-url'[, delimiter='char' ] | bucket='objecturl' )
```

Parameters

<code>url[, delimiter='char']</code>	<p>Specifies the URLs of one or more S3 objects to import, either the HTTPS URL or the S3 URL.</p> <p>URLs should contain only ASCII characters, 0x01 to 0x7F.</p> <p>If you specify multiple URLs, you can optionally qualify this parameter by specifying a delimiter character, by default (vertical bar). Do not use one of the following characters:</p> <ul style="list-style-type: none">• : (colon)
--	---

	<ul style="list-style-type: none">• - (hyphen)• , (comma)• / (slash)• null
bucket	URL of the bucket when importing multiple files using glob expansion.

Privileges

Write privileges on the table you are copying to.

Examples

The following statement specifies to import multiple files. Object URLs are delimited by vertical bars, the default delimiter:

```
=> COPY exampleTable SOURCE s3(url='s3://exampleBucket/object1|S3://exampleBucket/object2');
```

Import all files in a bucket using glob expansion:

```
=> COPY table1 WITH SOURCE S3(bucket='S3://exampleBucket/*');
```

```
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/');
```

```
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket');
```

See Also

- [AWS_SET_CONFIG](#)
- [AWS_GET_CONFIG](#)
- [AWS Library Parameters](#)
- [Vertica AWS Library](#)

S3EXPORT

Exports data to an Amazon S3 bucket from your Vertica cluster.

If your bucket name contains a period, you must specify a file extension in your export syntax or enable the `prepend_hash` parameter in `s3export`.

Syntax

```
S3EXPORT( expression USING PARAMETERS { parameter=setting }[,...]
```

parameter=setting

```
url='object-url'  
delimiter='char'  
multipart='char'  
chunksize=integer  
record_terminator='char'  
from_charset='varchar'  
to_charset='varchar'  
prepend_hash='boolean'
```

Parameters

<i>expression</i>	Specifies the source of the export operation.
<code>url</code>	The URL of the S3 bucket and object base name. This value can be either the HTTPS URL or the S3 URL. URL length is limited to a maximum of 236 characters. URLs should contain only ASCII characters, 0x01 to 0x7F.
<code>delimiter</code>	Specifies the column delimiter character. Default value: (vertical bar)
<code>multipart</code>	Specifies a Boolean value to enable (true) or disable (false) multipart upload. Default value: true
<code>chunksize</code>	Specifies in bytes the maximum size of each part. Valid settings can range between 5 MB and 5 GB. The maximum number of chunks allowed in an export is 10000. Default value: 10485760

<code>record_terminator</code>	Specifies what character marks the end of a record. Default value: <code>\n</code>
<code>from_charset</code>	Specifies the character set in which your data is currently encoded.
<code>to_charset</code>	Specifies the character set in which you want to encode your export.
<code>prepend_hash</code>	Prepends the unique hash values assigned to exported objects instead of the standard appendation. If your S3 bucket contains a period in its path, set the <code>prepend_hash</code> parameter to <code>true</code> . Default value: <code>false</code>

`from_charset` and `to_charset` values are system-dependent. Refer to your operating system documentation for more details.

Examples

Export `column1` data from `exampleTable`:

```
=> SELECT s3export(column1 USING PARAMETERS
  url='s3://exampleBucket/object',
  delimiter=',',
  multipart='true',
  chunksize='10485760',
  record_terminator='\n',
  from_charset='ASCII',
  to_charset='UTF-8',
  prepend_hash='true')
OVER () FROM exampleTable;
```

See Also

- [AWS_SET_CONFIG](#)
- [AWS_GET_CONFIG](#)
- [AWS Library Parameters](#)
- [Vertica AWS Library](#)

S3EXPORT_PARTITION

The S3EXPORT_PARTITION function allows Vertica output to be used by Amazon's Elastic MapReduce (EMR) feature. Since EMR stores and consumes data from S3 using the partition key included in the key of the S3 file, S3EXPORT_PARTITION exports data by adding the partition key in the url/filename.

Syntax

S3EXPORT_PARTITION (*expression* USING PARAMETERS { *parameter=setting*} [,...])

<i>expression</i>	Specifies the source of the export operation.
<i>url</i>	<p>The URL of the S3 bucket and object base name. Also include the partition key as part of the URL to export data so it is usable by EMR.</p> <p>The URL can be either the HTTPS URL or the S3 URL. URL length is limited to a maximum of 236 characters.</p> <p>URLs should contain only ASCII characters, 0x01 to 0x7F.</p>
<i>delimiter</i>	<p>Specifies the column delimiter character.</p> <p>Default value: (vertical bar)</p>
<i>chunksize</i>	<p>Determines the buffer size used to send bytes to S3. Valid settings can range between 5 MB and 5 GB.</p> <p>The maximum number of chunks allowed in an export is 10000.</p> <p>Default value: 10485760</p>
<i>record_terminator</i>	<p>Specifies what character marks the end of a record.</p> <p>Default value: \n</p>
<i>from_charset</i>	Specifies the character set in which your data is currently encoded.
<i>to_charset</i>	Specifies the character set in which you want to encode your export.
<i>prepend_hash</i>	Prepends the unique hash values assigned to exported objects instead of the standard appendation.

	If your S3 bucket contains a period in its path, set the <code>prepend_hash</code> parameter to true. Default value: false
<i>expression</i>	Specifies the source of the export operation.

`from_charset` and `to_charset` values are system-dependent. Refer to your operating system documentation for more details.

Examples

In the following example, `st` and `yr` are the partition keys:

```
=> select s3export_partition (* using parameters url='https://s3.amazonaws.com/db001/bystate.date;')
OVER (PARTITION by st, yr) from T;

rows | url
-----
184647| https://s3.amazonaws.com/db001/st=MA/yr=2005/bystate.77fcab9836b93a04.dat
282633| https://s3.amazonaws.com/db001/st=VA/yr=2007/bystate.77fcab9836b93a05.dat
282633| https://s3.amazonaws.com/db001/st=VA/yr=2009/bystate.77fcab9836b93a05.dat
(3 rows)
```

See Also

- [AWS_SET_CONFIG](#)
- [AWS_GET_CONFIG](#)
- [AWS Library Parameters](#)
- [Vertica AWS Library](#)

Catalog Management Functions

This section contains catalog management functions specific to Vertica.

DROP_LICENSE

Drops a license key from the global catalog.

Syntax

```
DROP_LICENSE( 'license-name' )
```

Parameters

<i>license-name</i>	The name of the license to drop. Use the name (or long license key) in the NAME column of system table LICENSES .
---------------------	---

Privileges

Superuser

Examples

```
=> SELECT DROP_LICENSE( '9b2d81e2-aab1-4cfb-bc07-fa9a696e8f5e' );
```

See Also

[Managing Licenses](#)

DUMP_CATALOG

Returns an internal representation of the Vertica catalog. This function is used for diagnostic purposes.

DUMP_CATALOG returns only the objects that are visible to the user.

Syntax

```
DUMP_CATALOG()
```

Privileges

None

Examples

The following query obtains an internal representation of the Vertica catalog:

```
=> SELECT DUMP_CATALOG();
```

The output is written to the specified file:

```
\o /tmp/catalog.txtSELECT DUMP_CATALOG();  
\o
```

EXPORT_CATALOG

Generates a SQL script for recreating a physical schema design on another cluster. This function always attempts to recreate projection statements with KSAFE clauses, if they exist in the original definitions, or OFFSET clauses if they do not.

Syntax

```
EXPORT_CATALOG ( '[destination]' [, 'scope' ] )
```

Parameters

If you omit all parameters, EXPORT_CATALOG exports to standard output all schemas, tables, constraints, views, and projections to which the user has access.

<i>destination</i>	Specifies where to send output, one of the following: <ul style="list-style-type: none">• An empty string (' ') writes the script to standard output.• The path and name of a SQL output file. This option is valid only for superusers. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.
<i>scope</i>	Determines what to export:

	<ul style="list-style-type: none">• DESIGN (default): Exports schemas, tables, constraints, views, projections, and SQL macros to which the user has access. See also EXPORT_OBJECTS.• DESIGN_ALL: Exports all design objects plus system objects created in Database Designer—for example, design contexts and their tables. Exported objects are those to which the user has access.• TABLES: Exports all tables and constraints for which the user has access. See also EXPORT_TABLES.• DIRECTED_QUERIES: Exports all directed queries that are stored in the database. For more information, see Managing Directed Queries in the Administrator's Guide.
--	---

Privileges

None

Example

See [Exporting the Catalog](#).

See Also

- [EXPORT_OBJECTS](#)
- [EXPORT_TABLES](#)
- [Exporting the Catalog](#)

EXPORT_OBJECTS

Generates a SQL script you can use to recreate non-virtual catalog objects on another cluster. The following requirements apply:

- **EXPORT_OBJECTS** only exports objects to which the user has access.

- EXPORT_OBJECTS exports objects in order dependency for correct recreation. When you run the script on another cluster, Vertica creates all referenced objects before their dependent objects.
- EXPORT_OBJECTS always tries to recreate projection statements with their KSAFE clause, if any, otherwise with their OFFSET clause.

Syntax

```
EXPORT_OBJECTS( '[destination]' [, 'scope' ] [, 'ksafe' ] )
```

Parameters

<i>destination</i>	<p>Specifies where to send output, one of the following:</p> <ul style="list-style-type: none"> • An empty string (' ') writes the script to standard output. • The path and name of a SQL output file. This option is valid only for superusers. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.
<i>scope</i>	<p>Specifies one or more objects to export, as follows:</p> <p><i>schema</i>[.<i>object</i>][, ...]</p> <p>If set to an empty string, Vertica exports all objects to which the user has access, including constraints.</p> <p>If you specify a schema, Vertica exports all objects in that schema.</p>
<i>ksafe</i>	<p>Specifies whether to include a MARK_DESIGN_KSAFE statement in the generated script with the correct K-safe value for the database:</p> <ul style="list-style-type: none"> • true (default): Include the MARK_DESIGN_KSAFE statement at the end of the output script. • false: Omit the MARK_DESIGN_KSAFE statement from the script.

Privileges

None

Example

See [Exporting Objects](#).

See Also

- [EXPORT_CATALOG](#)
- [EXPORT_TABLES](#)

EXPORT_TABLES

Generates a SQL script that can be used to recreate a logical schema—schemas, tables, constraints, and views—on another cluster. `EXPORT_OBJECTS` only exports objects to which the user has access.

Syntax

```
EXPORT_TABLES( '[destination]' [, 'scope' ] )
```

Parameters

<i>destination</i>	<p>Specifies where to send output, one of the following:</p> <ul style="list-style-type: none">• An empty string (' ') writes the script to standard output.• The path and name of a SQL output file. This option is valid only for superusers. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.
<i>scope</i>	<p>Specifies one or more tables to export, as follows:</p> <p><i>schema</i>[.<i>table</i>][,...]</p> <p>If set to an empty string, Vertica exports all non-virtual table objects to which the user has access, including table schemas, sequences, and constraints.</p> <p>If you specify a schema, Vertica exports all non-virtual table objects in that schema.</p>

Privileges

None

Example

See [Exporting Tables](#).

The following example exports to standard output all tables in the store schema:

```
=> SELECT EXPORT_TABLES('', 'store');
                                EXPORT_TABLES
-----
CREATE SCHEMA store;

CREATE TABLE store.store_dimension
(
  store_key int NOT NULL,
  store_name varchar(64),
  store_number int,
  store_address varchar(256),
  store_city varchar(64),
  store_state char(2),
  store_region varchar(64),
  floor_plan_type varchar(32),
  photo_processing_type varchar(32),
  financial_service_type varchar(32),
  selling_square_footage int,
  total_square_footage int,
  first_open_date date,
  last_remodel_date date,
  number_of_employees int,
  annual_shrinkage int,
  foot_traffic int,
  monthly_rent_cost int
);

ALTER TABLE store.store_dimension ADD CONSTRAINT C_PRIMARY PRIMARY KEY (store_key) DISABLED;

CREATE TABLE store.store_sales_fact
(
  date_key int NOT NULL,
  product_key int NOT NULL,
  product_version int NOT NULL,
  store_key int NOT NULL,
  promotion_key int NOT NULL,
  customer_key int NOT NULL,
  employee_key int NOT NULL,
  pos_transaction_number int NOT NULL,
  sales_quantity int,
  sales_dollar_amount int,
  cost_dollar_amount int,
  gross_profit_dollar_amount int,
  transaction_type varchar(16),
  transaction_time time,
```

```
    tender_type varchar(8)
);

CREATE TABLE store.store_orders_fact
(
    product_key int NOT NULL,
    product_version int NOT NULL,
    store_key int NOT NULL,
    vendor_key int NOT NULL,
    employee_key int NOT NULL,
    order_number int NOT NULL,
    date_ordered date,
    date_shipped date,
    expected_delivery_date date,
    date_delivered date,
    quantity_ordered int,
    quantity_delivered int,
    shipper_name varchar(32),
    unit_price int,
    shipping_cost int,
    total_order_cost int,
    quantity_in_stock int,
    reorder_level int,
    overstock_ceiling int
);

ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_date FOREIGN KEY (date_key)
references public.date_dimension (date_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_product FOREIGN KEY (product_key,
product_version) references public.product_dimension (product_key, product_version);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_store FOREIGN KEY (store_key)
references store.store_dimension (store_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_promotion FOREIGN KEY (promotion_
key) references public.promotion_dimension (promotion_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_customer FOREIGN KEY (customer_key)
references public.customer_dimension (customer_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_employee FOREIGN KEY (employee_key)
references public.employee_dimension (employee_key);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_product FOREIGN KEY (product_key,
product_version) references public.product_dimension (product_key, product_version);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_store FOREIGN KEY (store_key)
references store.store_dimension (store_key);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_vendor FOREIGN KEY (vendor_key)
references public.vendor_dimension (vendor_key);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_employee FOREIGN KEY (employee_
key) references public.employee_dimension (employee_key);

(1 row)
```

See Also

- [EXPORT_CATALOG](#)
- [EXPORT_OBJECTS](#)

INSTALL_LICENSE

Installs the license key in the global catalog.

Syntax

```
INSTALL_LICENSE( 'filename' )
```

Parameters

<i>filename</i>	The absolute path name of a valid license file.
-----------------	---

Privileges

Superuser

Examples

```
=> SELECT INSTALL_LICENSE('/tmp/vlicense.dat');
```

See Also

[Managing Licenses](#)

MARK_DESIGN_KSAFE

Enables or disables high availability in your environment, in case of a failure. Before enabling recovery, MARK_DESIGN_KSAFE queries the catalog to determine whether a cluster's physical schema design meets the following requirements:

- Small, unsegmented tables are replicated on all nodes.
- Large table superprojections are segmented with each segment on a different node.
- Each large table projection has at least one buddy projection for K-safety=1 (or two buddy projections for K-safety=2).

Buddy projections are also segmented across database nodes, but the distribution is modified so segments that contain the same data are distributed to different nodes. See [High Availability With Projections](#) in Vertica Concepts.

MARK_DESIGN_KSAFE does not change the physical schema.

Syntax

```
MARK_DESIGN_KSAFE ( k )
```

Parameters

<i>k</i>	<p>Specifies the level of K-safety, one of the following:</p> <ul style="list-style-type: none">• 2: Enables high availability if the schema design meets requirements for K-safety=2• 1: Enables high availability if the schema design meets requirements for K-safety=1• 0: Disables high availability
----------	---

Return Messages

If you specify a *k* value of 1 or 2, Vertica returns one of the following messages.

Success:

```
Marked design n-safe
```

Failure:

```
The schema does not meet requirements for K=n.  
Fact table projection projection-name  
has insufficient "buddy" projections.
```

where *n* is a K-safety setting.

Privileges

Superuser

Notes

- The database's internal recovery state persists across database restarts but it is not checked at startup time.
- When one node fails on a system marked K-safe=1, the remaining nodes are available for DML operations.

Examples

```
=> SELECT MARK_DESIGN_KSAFE(1);
   mark_design_ksafe
-----
Marked design 1-safe
(1 row)
```

If the physical schema design is not K-safe, messages indicate which projections do not have a buddy:

```
=> SELECT MARK_DESIGN_KSAFE(1);
The given K value is not correct;
the schema is 0-safe
Projection pp1 has 0 buddies,
which is smaller that the given K of 1
Projection pp2 has 0 buddies,
which is smaller that the given K of 1
.
.
.
(1 row)
```

See Also

- [Identical Segmentation](#)
- [Failure Recovery](#)

CFS Security Functions

This section describes functions you can use with the Connector Framework Services (CFS) security features.

These functions reside in the `v_idol` schema in the `idollib` library that is installed with CFS. When you run one of these functions, you must qualify its name with the `v_idol` schema. For example:

```
=> SELECT v_idol.DELETE_COMMUNITY_KEY();
```

DELETE_COMMUNITY_KEY

Removes the security key from the Vertica database DFS.

Syntax

```
V_IDOL.DELETE_COMMUNITY_KEY()
```

Privileges

Superuser

Examples

```
=> SELECT V_IDOL.DELETE_COMMUNITY_KEY();  
DELETE_COMMUNITY_KEY  
-----  
key successfully deleted
```

DESCRIBE_COMMUNITY_KEY

Retrieves the value of the security key so you can use it in a query.

Syntax

```
V_IDOL.DESCRIBE_COMMUNITY_KEY()
```

Example

This example retrieves the security key stored in the the Vertica database DFS:

```
=> SELECT V_IDOL.DESCRIBE_COMMUNITY_KEY();  
describe_community_key  
-----
```

```
123.144,564,231  
(1 row)
```

IDOL_CHECK_ACL

Checks the access control list to verify that the user has permissions to access the data in the IDOL flex table and any views created from the table.

Syntax

```
v_idol.idol_check_acl(autonomymetadata,securitysection, securitytype using parameters  
sis='MTMyfNudFzPXVB5rX2dLnU89CQo+B668H6iNiHr14KPC+ptoA4x80KzeFmme/y3QNYzQs+QVMd7nmvPc4CaT9qxNr3t2EMgr8YN  
GJ5KKnE61YN4BixkKd241tEN1LZ08c  
CJXiQaTppIqayD8UI9aC+JVvtGxeyc003cqrVQiMMqxzmrHEQfw==' ) from v_idol.idolsecurity;
```

You can also use the session parameter as part of the syntax:

```
Alter session set UDPARAMETER FOR v_idol.IdolLib IdolSecurityInfo ='MTMyf ... <encrypted SIS> ...BCmky';  
SELECT v_idol.idol_check_acl(acl, security_section, security_type) from v_idol.t;
```

Privileges

To run `idol_check_acl` on a view, you must be granted access to the view by the database administration user. This example creates a table, creates a view from the table, and grants access to the view to user1:

```
=> CREATE TABLE idoldata_base(...);  
=> CREATE VIEW idoldata as select * from idoldata_base where idol_  
check_acl(...);  
=> GRANT SELECT ON idoldata to user1;
```

Returns

This function returns one of the following:

- t - indicates the user has permission to access the data.
- f - indicates the user does not have permission to access the data.

INSTALL_COMMUNITY_KEY

Stores the Connector Framework Service security key in the Vertica Distributed File System (DFS).

Syntax

```
V_IDOL.INSTALL_COMMUNITY_KEY(USING PARAMETERS file_path='keyfile-path');
```

Parameters

file_path	Specifies the location of key file from which the key is copied to DFS.
-----------	---

Privileges

You must be a dbadmin user.

Example

Store the CFS security key that is in btea . key:

```
=> SELECT v_idol.install_community_key(  
    USING PARAMETERS file_path='/home/release/IDOL/test-data/btea.key');
```

Client Connection Management Functions

This section contains client connection management functions specific to Vertica.

GET_CLIENT_LABEL

Returns the client connection label for the current session.

Syntax

```
GET_CLIENT_LABEL()
```

Privileges

None

Examples

Return the current client connection label:

```
=> SELECT GET_CLIENT_LABEL();
      GET_CLIENT_LABEL
-----
data_load_application
(1 row)
```

See Also

[Setting a Client Connection Label](#)

RESET_LOAD_BALANCE_POLICY

Resets the counter each host in the cluster maintains, to track which host it will refer a client to when the native [connection load balancing scheme](#) is set to ROUNDROBIN. To reset the counter, run this function on all cluster nodes.

Syntax

```
RESET_LOAD_BALANCE_POLICY()
```

Privileges

Superuser

Example

```
=> SELECT RESET_LOAD_BALANCE_POLICY();
      RESET_LOAD_BALANCE_POLICY
-----
Successfully reset stateful client load balance policies: "roundrobin".
(1 row)
```

SET_CLIENT_LABEL

Assigns a label to a client connection for the current session. You can use this label to distinguish client connections.

Labels appear in the `v_monitor.sessions` table. However, they are not updated in Data Collector tables because the change occurs after Vertica makes the connection.

Syntax

```
SET_CLIENT_LABEL('Label-name')
```

Parameters

<i>Label-name</i>	VARCHAR name assigned to the client connection label.
-------------------	---

Privileges

None

Examples

Assign label `data_load_application` to the current client connection:

```
=> SELECT SET_CLIENT_LABEL('data_load_application');
       SET_CLIENT_LABEL
-----
client_label set to data_load_application
(1 row)
```

See Also

[Setting a Client Connection Label](#)

SET_LOAD_BALANCE_POLICY

Sets how native connection load balancing chooses a host to handle a client connection.

Syntax

```
SET_LOAD_BALANCE_POLICY(policy)
```

Parameters

<i>policy</i>	<p>The name of the load balancing policy to use, one of the following:</p> <ul style="list-style-type: none">• NONE (default): Disables native connection load balancing.• ROUNDROBIN: Chooses the next host from a circular list of hosts in the cluster that are up—for example, in a three-node cluster, iterates over node1, node2, and node3, then wraps back to node1. Each host in the cluster maintains its own pointer to the next host in the circular list, rather than there being a single cluster-wide state.• RANDOM: Randomly chooses a host from among all hosts in the cluster that are up. <p>Note: Even if the load balancing policy is set on the server to something other than NONE, clients must indicate they want their connections to be load balanced by setting a connection property.</p>
---------------	---

Privileges

Superuser

Example

The following example demonstrates enabling native connection load balancing on the server by setting the load balancing scheme to ROUNDROBIN:

```
=> SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');
          SET_LOAD_BALANCE_POLICY
-----
Successfully changed the client initiator load balancing policy to: roundrobin
(1 row)
```

See Also

[About Native Connection Load Balancing](#)

Cluster Management Functions

This section contains functions that manage spread deployment on large, distributed database clusters.

REALIGN_CONTROL_NODES

Chooses control nodes (spread hosts) from all cluster nodes and assigns the rest of the nodes in the cluster to a control node. Calling this function respects existing [fault groups](#), which you can view by querying the [V_CATALOG.CLUSTER_LAYOUT](#) system table. This view also lets you see the proposed new layout for nodes in the cluster.

Syntax

```
REALIGN_CONTROL_NODES()
```

Privileges

Superuser

Example

Choose control nodes from all cluster nodes and assign the remaining nodes to a control node:

```
=> SELECT REALIGN_CONTROL_NODES;
```

See Also

[Defining and Realigning Control Nodes on an Existing Cluster](#)

REBALANCE_CLUSTER

Rebalances the database cluster synchronously as a session foreground task. `REBALANCE_CLUSTER` returns only after the rebalance operation is complete. If the current session ends, the operation immediately aborts. To rebalance the cluster as a background task, call [START_REBALANCE_CLUSTER](#).

On large cluster arrangements, you typically call `REBALANCE_CLUSTER` in a flow (see [Defining and Realigning Control Nodes](#) in the Administrator's Guide). After you change the number and distribution of control nodes (spread hosts), you must run `REBALANCE_CLUSTER()` for fault tolerance to be realized.

For detailed information about rebalancing tasks, see [Rebalancing Data Across Nodes](#).

Tip: By default, before performing a rebalance, Vertica queries system tables to compute the size of all projections involved in the rebalance task. This query can add significant overhead to the rebalance operation. To disable this query, set projection configuration parameter `RebalanceQueryStorageContainers` to 0.

Syntax

```
REBALANCE_CLUSTER()
```

Privileges

Superuser

Example

```
=> SELECT REBALANCE_CLUSTER();
REBALANCE_CLUSTER
-----
REBALANCED
(1 row)
```

RELOAD_SPREAD

Updates cluster changes to the catalog's spread configuration file. These changes include:

- New or realigned control nodes
- New spread hosts or fault group
- New or dropped cluster nodes

This function is often used in a multi-step process for large and elastic cluster arrangements. Calling it might require you to restart the database. You must then rebalance the cluster to realize fault tolerance. For details, see [Defining and Realigning Control Nodes](#) in the Administrator's Guide.

Syntax

```
RELOAD_SPREAD( true )
```

Parameters

<i>true</i>	Updates cluster changes related to control message responsibilities to the spread configuration file.
-------------	---

Privileges

Superuser

Example

Update the cluster with changes to control messaging:

```
=> SELECT reload_spread(true);
 reload_spread
-----
reloaded
(1 row)
```

See Also

[REBALANCE_CLUSTER](#)

SET_CONTROL_SET_SIZE

Specifies the number of cluster nodes on which to deploy control messaging (spread). You can call this function only on a running database.

To determine whether the current spread hosts and the control designations in the catalog match, query system table [LARGE_CLUSTER_CONFIGURATION_STATUS](#)

Note: This function is equivalent to:

```
install_vertica --large cluster integer
```

For details, see [Installing a Large Cluster](#) in the Administrator's Guide.

Syntax

```
SET_CONTROL_SET_SIZE( integer )
```

Parameters

<i>integer</i>	The number of cluster hosts from the database cluster on which spread runs.
----------------	---

Privileges

Superuser

Example

Run spread on two cluster nodes:

```
=> SELECT set_control_set_size(2);
   SET_CONTROL_SET_SIZE
-----
Control size set
(1 row)
```

See Also

[Defining and Realigining Control Nodes on an Existing Cluster](#)

Cluster Scaling Functions

This section contains functions that control how the cluster organizes data for rebalancing.

CANCEL_REBALANCE_CLUSTER

Stops any rebalance task that is currently in progress or is waiting to execute.

Syntax

```
CANCEL_REBALANCE_CLUSTER()
```

Privileges

Superuser

Example

```
=> SELECT CANCEL_REBALANCE_CLUSTER();
CANCEL_REBALANCE_CLUSTER
-----
CANCELED
(1 row)
```

See Also

- [START_REBALANCE_CLUSTER](#)
- [REBALANCE_CLUSTER](#)

DISABLE_ELASTIC_CLUSTER

Disables elastic cluster scaling, which prevents Vertica from bundling data into chunks that are easily transportable to other nodes when performing cluster resizing. The main reason to disable elastic clustering is if you find that the slightly unequal data distribution in your cluster caused by grouping data into discrete blocks results in performance issues.

Syntax

```
DISABLE_ELASTIC_CLUSTER()
```

Privileges

Superuser

Example

```
=> SELECT DISABLE_ELASTIC_CLUSTER();
DISABLE_ELASTIC_CLUSTER
-----
DISABLED
(1 row)
```

See Also

- [ENABLE_ELASTIC_CLUSTER](#)

DISABLE_LOCAL_SEGMENTS

Disables local data segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See [Local Data Segmentation](#) in the Administrator's Guide for details.

Syntax

```
DISABLE_LOCAL_SEGMENTS()
```

Privileges

Superuser

Example

```
=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
-----
DISABLED
```

```
(1 row)
```

ENABLE_ELASTIC_CLUSTER

Enables elastic cluster scaling, which makes enlarging or reducing the size of your database cluster more efficient by segmenting a node's data into chunks that can be easily moved to other hosts.

Syntax

```
ENABLE_ELASTIC_CLUSTER()
```

Privileges

Superuser

Example

```
=> SELECT ENABLE_ELASTIC_CLUSTER();
ENABLE_ELASTIC_CLUSTER
-----
ENABLED
(1 row)
```

See Also

- [DISABLE_ELASTIC_CLUSTER](#)

ENABLE_LOCAL_SEGMENTS

Enables local storage segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See [Local Data Segmentation](#) in the Administrator's Guide for more information.

Syntax

```
ENABLE_LOCAL_SEGMENTS()
```

Privileges

Superuser

Example

```
=> SELECT ENABLE_LOCAL_SEGMENTS();
   ENABLE_LOCAL_SEGMENTS
-----
   ENABLED
(1 row)
```

SET_SCALING_FACTOR

Sets the scaling factor that determines the number of storage containers used when rebalancing the database and when using local data segmentation is enabled. See [Cluster Scaling](#) for details.

Syntax

```
SET_SCALING_FACTOR( factor )
```

Parameters

<i>factor</i>	An integer value between 1 and 32. Vertica uses this value to calculate the number of storage containers each projection is broken into when rebalancing or when local data segmentation is enabled.
---------------	--

Privileges

Superuser

Best Practices

The scaling factor determines the number of storage containers that Vertica uses to store each projection across the database during rebalancing when local segmentation is enabled. When setting the scaling factor, follow these guidelines:

- The number of storage containers should be greater than or equal to the number of partitions multiplied by the number of local segments:

$$\text{num-storage-containers} \geq (\text{num-partitions} * \text{num-local-segments})$$

- Set the scaling factor high enough so rebalance can transfer local segments to satisfy the skew threshold, but small enough so the number of storage containers does not result in too many ROS containers, and cause ROS *pushback*. The maximum number of ROS containers is 1024.

Example

```
=> SELECT SET_SCALING_FACTOR(12);
   SET_SCALING_FACTOR
-----
SET
(1 row)
```

START_REBALANCE_CLUSTER

Asynchronously rebalances the database cluster as a background task. This function returns immediately after the rebalancing operation is complete. Rebalancing persists until the operation is complete, even if you close the current session or the database shuts down. In the case of shutdown, rebalancing resumes after the cluster restarts. To stop the rebalance operation, call [CANCEL_REBALANCE_CLUSTER](#).

For detailed information about rebalancing tasks, see [Rebalancing Data Across Nodes](#).

Syntax

```
START_REBALANCE_CLUSTER()
```

Privileges

Superuser

Example

```
=> SELECT START_REBALANCE_CLUSTER();
   START_REBALANCE_CLUSTER
-----
```

```
REBALANCING  
(1 row)
```

See Also

[REBALANCE_CLUSTER](#)

Communications Functions

This section contains communication functions specific to Vertica.

NOTIFY

Specifies the text message to include with a notification.

Syntax

```
NOTIFY ( 'message', 'notifier', 'target-topic' )
```

Parameters

<i>message</i>	The message to send to the end point.
<i>notifier</i>	The name of the notifier used to deliver the message.
<i>target-topic</i>	The name of the destination Kafka topic for the message.

Privileges

Superuser

Examples

Send a message to confirm that an ETL job is complete:

```
=> SELECT NOTIFY('ETL Done!', 'my_notifier', 'DB_activity_topic');
```

Constraint Management Functions

This section contains constraint management functions specific to Vertica.

See also SQL system table [V_CATALOG.TABLE_CONSTRAINTS](#).

ANALYZE_CONSTRAINTS

Analyzes and reports on constraint violations within the specified scope

You can enable automatic enforcement of primary key, unique key, and check constraints when INSERT, UPDATE, MERGE, or COPY statements execute. Alternatively, you can use ANALYZE_CONSTRAINTS to validate constraints after issuing these statements. Refer to [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#) for more information.

ANALYZE_CONSTRAINTS performs a lock in the same way that SELECT * FROM t1 holds a lock on table t1. See [LOCKS](#) for additional information.

Syntax

```
ANALYZE_CONSTRAINTS ('[ schema.]table ' [, 'column[,...]')
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>table-name</i>	Identifies the table to analyze. If you omit specifying a schema, Vertica uses the current schema search path. If set to an empty string, Vertica analyzes all tables in the current schema.
<i>column</i>	The column in <i>table</i> to analyze. You can specify multiple comma-delimited columns. Vertica narrows the scope of the analysis to the specified columns. If you omit specifying a column, Vertica analyzes all columns in <i>table</i> .

Privileges

- SELECT privilege on table
- USAGE privilege on schema

Detecting Constraint Violations During a Load Process

Vertica checks for constraint violations when queries are run, not when data is loaded. To detect constraint violations as part of the load process, use a [COPY](#) statement with the NO COMMIT option. By loading data without committing it, you can run a post-load check of your data using the ANALYZE_CONSTRAINTS function. If the function finds constraint violations, you can roll back the load because you have not committed it.

If ANALYZE_CONSTRAINTS finds violations, such as when you insert a duplicate value into a primary key, you can correct errors using the following functions. Effects last until the end of the session only:

- [DISABLE_DUPLICATE_KEY_ERROR](#)
- [REENABLE_DUPLICATE_KEY_ERROR](#)

Important: If a check constraint SQL expression evaluates to an unknown for a given row because a column within the expression contains a null, the row passes the constraint condition.

Return Values

ANALYZE_CONSTRAINTS returns results in a structured set (see table below) that lists the schema name, table name, column name, constraint name, constraint type, and the column values that caused the violation.

If the result set is empty, then no constraint violations exist; for example:

```
> SELECT ANALYZE_CONSTRAINTS ('public.product_dimension', 'product_key');
Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
(0 rows)
```

The following result set shows a primary key violation, along with the value that caused the violation ('10'):

```
=> SELECT ANALYZE_CONSTRAINTS ('');
Schema Name | Table Name | Column Names | Constraint Name | Constraint Type | Column Values
-----+-----+-----+-----+-----+-----
store       | t1         | c1           | pk_t1          | PRIMARY        | ('10')
```

(1 row)

The result set columns are described in further detail in the following table:

Column Name	Data Type	Description
Schema Name	VARCHAR	The name of the schema.
Table Name	VARCHAR	The name of the table, if specified.
Column Names	VARCHAR	A list of comma-delimited columns that contain constraints.
Constraint Name	VARCHAR	The given name of the primary key, foreign key, unique, check, or not null constraint, if specified.
Constraint Type	VARCHAR	Identified by one of the following strings: <ul style="list-style-type: none"> • PRIMARY KEY • FOREIGN KEY • UNIQUE • CHECK • NOT NULL
Column Values	VARCHAR	Value of the constraint column, in the same order in which Column Names contains the value of that column in the violating row. When interpreted as SQL, the value of this column forms a list of values of the same type as the columns in Column Names; for example: ('1'), ('1', 'z')

Examples

See [Detecting Constraint Violations with ANALYZE_CONSTRAINTS](#) in the Administrator's Guide.

ANALYZE_CORRELATIONS

Analyzes the specified tables for pairs of columns that are strongly correlated. ANALYZE_CORRELATIONS stores the 20 pairs with the strongest correlation. ANALYZE_CORRELATIONS also analyzes statistics.

ANALYZE_CORRELATIONS analyzes only pairwise single-column correlations.

For example, state name and country name columns are strongly correlated because the city name usually, but perhaps not always, identifies the state name. The city of Conshohocken is uniquely associated with Pennsylvania, while the city of Boston exists in Georgia, Indiana, Kentucky, New York, Virginia, and Massachusetts. In this case, city name is strongly correlated with state name.

Behavior Type

Immutable

Syntax

```
ANALYZE_CORRELATIONS ('[ [schema.]table ]' [, 'recalculate' ] )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	<p>Identifies the table to analyze. If you omit specifying a schema, Vertica uses the current schema search path. If set to an empty string, Vertica analyzes all tables in the current schema.</p>
<i>recalculate</i>	<p>Boolean that specifies whether to analyze correlated columns that were previously analyzed.</p> <p>Note: Column correlation analysis typically needs to be done only once.</p> <p>Default: false</p>

Privileges

One of the following:

- Superuser
- User with USAGE privilege on the design schema

Analysis Follow-Up

To take advantage of the correlations that ANALYZE_CORRELATIONS discovers, run Database Designer programmatically. Run [DESIGNER_SET_ANALYZE_CORRELATIONS_MODE](#) to specify that Database Designer consider existing column correlations. Be sure to specify that Database Designer not analyze statistics so it does not override existing statistics.

Example

In the following example, ANALYZE_CORRELATIONS analyzes column correlations for all tables in the public schema, even if they currently exist. The correlations that ANALYZE_CORRELATIONS finds are saved, so Database Designer can use them the next time it runs on the VMart database:

```
=> SELECT ANALYZE_CORRELATIONS ('public.*', 'true');
ANALYZE_CORRELATIONS
-----
0
(1 row)
```

DISABLE_DUPLICATE_KEY_ERROR

Disables error messaging when Vertica finds duplicate primary or unique key values at run time (for use with key constraints that are not automatically enabled). Queries execute as though no constraints are defined on the schema. Effects are session scoped.

Syntax

```
DISABLE_DUPLICATE_KEY_ERROR();
```

Privileges

Superuser

Examples

When you call `DISABLE_DUPLICATE_KEY_ERROR`, Vertica issues warnings letting you know that duplicate values will be ignored, and incorrect results are possible. `DISABLE_DUPLICATE_KEY_ERROR` is for use only for key constraints that are not automatically enabled.

```
=> select DISABLE_DUPLICATE_KEY_ERROR();
WARNING 3152: Duplicate values in columns marked as UNIQUE will now be ignored for the remainder of
your session or until reenable_duplicate_key_error() is called
WARNING 3539: Incorrect results are possible. Please contact Vertica Support if unsure
  disable_duplicate_key_error
-----
Duplicate key error disabled
(1 row)
```

See Also

[ANALYZE_CONSTRAINTS](#)

LAST_INSERT_ID

Returns the last value of a column whose value is automatically incremented through `AUTO_INCREMENT` or `IDENTITY` [column constraints](#). If multiple sessions concurrently load the same table, the returned value is the last value generated for an `AUTO_INCREMENT` column by an insert in that session.

Behavior Type

Volatile

Syntax

```
LAST_INSERT_ID()
```

Privileges

- Table owner
- `USAGE` privileges on schema

Restrictions

- This function works only with AUTO_INCREMENT and IDENTITY columns. See [column constraints](#) for [CREATE TABLE](#).
- LAST_INSERT_ID does not work with sequence generators created through [CREATE SEQUENCE](#).

Examples

1. Create table customer4:

```
=> CREATE TABLE customer4(  
    ID IDENTITY(2,2),  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER  
);  
=> INSERT INTO customer4(lname, fname, membership_card) VALUES ('Gupta', 'Saleem', 475987);
```

The IDENTITY column has a seed of 2, which specifies the value for the first row loaded into the table, and an increment of 2, which specifies the value that is added to the IDENTITY value of the previous row.

2. Query the table you just created:

```
=> SELECT * FROM customer4;  
ID | lname | fname | membership_card  
-----+-----+-----+-----  
  2 | Gupta | Saleem |          475987  
(1 row)
```

3. Insert additional values:

```
=> INSERT INTO customer4(lname, fname, membership_card) VALUES ('Lee', 'Chen', 598742);
```

4. Call LAST_INSERT_ID:

```
=> SELECT LAST_INSERT_ID();  
LAST_INSERT_ID  
-----  
                4  
(1 row)
```

5. Query the table again:

```
=> SELECT * FROM customer4;
ID | lname | fname | membership_card
-----+-----+-----+-----
 2 | Gupta | Saleem |          475987
 4 | Lee   | Chen   |          598742
(2 rows)
```

6. Add another row:

```
=> INSERT INTO customer4(lname, fname, membership_card) VALUES ('Davis', 'Bill', 469543);
```

7. Call LAST_INSERT_ID:

```
=> SELECT LAST_INSERT_ID();
LAST_INSERT_ID
-----
              6
(1 row)
```

8. Query the table again:

```
=> SELECT * FROM customer4;
ID | lname | fname | membership_card
-----+-----+-----+-----
 2 | Gupta | Saleem |          475987
 4 | Lee   | Chen   |          598742
 6 | Davis | Bill   |          469543
(3 rows)
```

See Also

- [Working with Sequence Types](#)
- [Sequence Privileges](#)

REENABLE_DUPLICATE_KEY_ERROR

Restores the default behavior of error reporting by reversing the effects of [DISABLE_DUPLICATE_KEY_ERROR](#). Effects are session scoped.

Syntax

```
REENABLE_DUPLICATE_KEY_ERROR();
```

Privileges

Superuser

Examples

```
=> SELECT REENABLE_DUPLICATE_KEY_ERROR();
reenable_duplicate_key_error
-----
Duplicate key error enabled
(1 row)
```

See Also

[ANALYZE_CONSTRAINTS](#)

Data Collector Functions

The Vertica Data Collector is a utility that extends [system table](#) functionality by providing a framework for recording events. It gathers and retains monitoring information about your database cluster and makes that information available in system tables, requiring few configuration parameter tweaks, and having negligible impact on performance.

Collected data is stored on disk in the `DataCollector` directory under the Vertica `/catalog` path. You can use the information the Data Collector retains to query the past state of system tables and extract aggregate information, as well as do the following:

- See what actions users have taken
- Locate performance bottlenecks
- Identify potential improvements to Vertica configuration

Data Collector works in conjunction with an advisor tool called Workload Analyzer, which intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history.

By default, Data Collector is on and retains information for all sessions. If performance issues arise, a superuser can disable DC. See [Data Collector Parameters](#) and [Enabling and Disabling Data Collector](#) in the Administrator's Guide.

This section describes the Data Collection control functions.

Related Topics

[V_MONITOR.DATA_COLLECTOR](#)

[Retaining Monitoring Information](#) and [Analyzing Workloads](#) in the Administrator's Guide

CLEAR_DATA_COLLECTOR

Clears all memory and disk records on the Data Collector tables and functions and resets collection statistics in the system table [DATA_COLLECTOR](#). A superuser can clear Data Collector data for all components or specify an individual component

After you clear the Data Collector log, the information is no longer available for querying.

Syntax

```
CLEAR_DATA_COLLECTOR( [ 'component' ] )
```

Parameters

<i>component</i>	<p>Clears memory and disk records for the specified component only. If you provide no argument, the function clears all Data Collector memory and disk records for all components.</p> <p>For the current list of component names, query the system table DATA_COLLECTOR.</p>
------------------	---

Privileges

Superuser

Examples

The following command clears memory and disk records for the ResourceAcquisitions component:

```
=> SELECT clear_data_collector('ResourceAcquisitions');
clear_data_collector
-----
CLEAR
(1 row)
```

The following command clears data collection for all components on all nodes:

```
=> SELECT clear_data_collector();
clear_data_collector
-----
CLEAR
(1 row)
```

See Also

- [DATA_COLLECTOR](#)
- [Retaining Monitoring Information](#)

DATA_COLLECTOR_HELP

Returns online usage instructions about the Data Collector, the [DATA_COLLECTOR](#) system table, and the Data Collector control functions.

Syntax

```
DATA_COLLECTOR_HELP()
```

Privileges

None

Returns

The `DATA_COLLECTOR_HELP()` function returns the following information:

```
=> SELECT DATA_COLLECTOR_HELP();

-----
Usage Data Collector
The data collector retains history of important system activities.
  This data can be used as a reference of what actions have been taken
  by users, but it can also be used to locate performance bottlenecks,
  or identify potential improvements to the Vertica configuration.
  This data is queryable via Vertica system tables.

Access a list of data collector components, and some statistics, by running:
SELECT * FROM v_monitor.data_collector;

The amount of data retained by size and time can be controlled with several
functions.
  To just set the size amount:
      set_data_collector_policy(<component>,
                              <memory retention (KB)>,
                              <disk retention (KB)>);

  To set both the size and time amounts (the smaller one will dominate):
      set_data_collector_policy(<component>,
                              <memory retention (KB)>,
                              <disk retention (KB)>,
                              <interval>);

  To set just the time amount:
      set_data_collector_time_policy(<component>,
                                     <interval>);

  To set the time amount for all tables:
      set_data_collector_time_policy(<interval>);
```

The current retention policy for a component can be queried with:
`get_data_collector_policy(<component>);`

Data on disk is kept in the "DataCollector" directory under the Vertica \catalog path. This directory also contains instructions on how to load the monitoring data into another Vertica database.

To move the data collector logs and instructions to other storage locations, create labeled storage locations using `add_location` and then use:

```
set_data_collector_storage_location(<storage_label>);
```

Additional commands can be used to configure the data collection logs.

The log can be cleared with:

```
clear_data_collector([<optional component>]);
```

The log can be synchronized with the disk storage using:

```
flush_data_collector([<optional component>]);
```

See Also

- [DATA_COLLECTOR](#)
- [TUNING_RECOMMENDATIONS](#)
- [Analyzing Workloads](#)
- [Retaining Monitoring Information](#)

FLUSH_DATA_COLLECTOR

Waits until memory logs are moved to disk and then flushes the Data Collector, synchronizing the log with the disk storage. A superuser can flush Data Collector information for an individual component or for all components.

Syntax

```
FLUSH_DATA_COLLECTOR( [ 'component' ] )
```

Parameters

<i>component</i>	Flushes the specified component. If you provide no argument, the function flushes the Data Collector in full. For the current list of component names, query the V_MONITOR.DATA_
------------------	---

	COLLECTOR system table.
--	-------------------------

Privileges

Superuser

Examples

The following command flushes the Data Collector for the ResourceAcquisitions component:

```
=> SELECT flush_data_collector('ResourceAcquisitions');
flush_data_collector
-----
FLUSH
(1 row)
```

The following command flushes data collection for all components:

```
=> SELECT flush_data_collector();
flush_data_collector
-----
FLUSH
(1 row)
```

See Also

- [DATA_COLLECTOR](#)
- [Retaining Monitoring Information](#)

GET_DATA_COLLECTOR_POLICY

Retrieves a brief statement about the retention policy for the specified component.

Syntax

```
GET_DATA_COLLECTOR_POLICY( 'component' )
```

Parameters

<i>component</i>	Returns the retention policy for the specified component.
------------------	---

	For a current list of component names, query the V_MONITOR.DATA_COLLECTOR system table
--	--

Privileges

None

Example

The following query returns the history of all resource acquisitions by specifying the ResourceAcquisitions component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
       get_data_collector_policy
-----
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```

See Also

- [DATA_COLLECTOR](#)
- [Retaining Monitoring Information](#)

SET_DATA_COLLECTOR_POLICY

Sets a size restraint (memory and disk space in kilobytes) for the specified Data Collector table on all nodes. If nodes are down, the failed nodes receive the setting when they rejoin the cluster.

You can use this function to set a size restraint only, or you can include the optional *interval* argument to set disk capacity for both size and time in a single command.

If you specify *interval*, Vertica enforces the setting that is exceeded first (size or time). Before you include a time restraint, verify that the disk size capacity is sufficiently large.

If you want to specify just a time restraint, or you want to turn off a time restraint you set using this function, see [SET_DATA_COLLECTOR_TIME_POLICY\(\)](#).

Syntax

```
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB' [, 'interval'] )
```

Parameters

<i>component</i>	Configures the retention policy for the specified component.
<i>memoryKB</i>	Specifies the memory size to retain in kilobytes.
<i>diskKB</i>	Specifies the disk size in kilobytes.
<i>interval</i>	[Default off] Takes an optional <i>interval</i> argument to specify how long to retain the specified component on disk. To disable a time restraint, set <i>interval</i> to -1. Note: Any negative input will turn off the time restraint

Privileges

Superuser

Notes

- Before you change a retention policy, view its current setting by calling the `GET_DATA_COLLECTOR_POLICY()` function.
- If you don't know the name of a component, query the `V_MONITOR.DATA_COLLECTOR` system table for a list; for example:

```
=> SELECT DISTINCT component, description FROM data_collector  
ORDER BY 1 ASC;
```

Examples

The following command returns the retention policy for the `ResourceAcquisitions` component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');  
get_data_collector_policy  
-----  
1000KB kept in memory, 10000KB kept on disk.  
(1 row)
```

This command changes the memory and disk setting for `ResourceAcquisitions` from its current setting of 1,000 KB memory and 10,000 KB disk space to 1500 KB and 25000 KB, respectively:

```
=> SELECT set_data_collector_policy('ResourceAcquisitions', '1500', '25000');
       set_data_collector_policy
-----
      SET
(1 row)
```

This command sets the `RequestsIssued` component to 1500 KB memory and 11000 KB on disk, and includes a 3-minute time restraint:

```
=> SELECT set_data_collector_policy('RequestsIssued', '1500', '11000', '3 minutes'::interval);
       set_data_collector_policy
-----
      SET
(1 row)
```

The following command disables the 3-minute retention policy for the `RequestsIssued` component, using [SET_DATA_COLLECTOR_TIME_POLICY](#):

```
=> SELECT set_data_collector_time_policy('RequestsIssued', '-1');
       set_data_collector_time_policy
-----
      SET
(1 row)
```

See Also

- [GET_DATA_COLLECTOR_POLICY](#)
- [SET_DATA_COLLECTOR_TIME_POLICY\(\)](#)
- [DATA_COLLECTOR](#)
- [Retaining Monitoring Information](#) in the *Administrator's Guide*

SET_DATA_COLLECTOR_TIME_POLICY

Sets a time capacity for individual Data Collector tables on all nodes. If nodes are down, the failed nodes receive the setting when they rejoin the cluster.

If you specify *interval*, Vertica enforces the setting that is exceeded first (size or time). Before you include a time restraint, verify that the disk size capacity is sufficiently large.

If you want to configure both time and size restraints at the same time, see [SET_DATA_COLLECTOR_POLICY](#).

Syntax

```
SET_DATA_COLLECTOR_TIME_POLICY( [component'], interval' )
```

Parameters

<i>component</i>	[Optional] Configures the time retention policy for the specified component. If you omit the <i>component</i> argument, Vertica sets the specified time capacity for all Data Collector tables.
<i>interval</i>	Specifies the time restraint on disk using an INTERVAL type. To disable a time restraint, set <i>interval</i> to -1. Note: Any negative input turns off the time restraint

Privileges

Superuser

Usage Considerations

- Before you change a retention policy, view its current setting by calling the [GET_DATA_COLLECTOR_POLICY](#) function.
- If you don't know the name of a component, query the V_MONITOR.DATA_COLLECTOR system table for a list. For example:

```
=> SELECT DISTINCT component, description FROM data_collector  
ORDER BY 1 ASC;
```

Setting the Time Interval for System Tables

You can also use the *interval* argument to query system tables the same way you query Data Collector tables; for example:

```
set_data_collector_time_policy('<system-table>', <'interval'>);
```

To illustrate, the following command in the left column is equivalent to running the series of commands on the right:

Run one command	Instead of a series of commands
<pre>SELECT set_data_collector_time_policy ('v_monitor.query_requests', '3 minutes'::interval);</pre>	<pre>SELECT set_data_collector_time_policy ('RequestsIssued', '3 minutes'::interval); SELECT set_data_collector_time_policy ('RequestsCompleted', '3 minutes'::interval); SELECT set_data_collector_time_policy ('Errors', '3 minutes'::interval); SELECT set_data_collector_time_policy ('ResourceAcquisitions', '3 minutes'::interval);</pre>

The [SET_DATA_COLLECTOR_TIME_POLICY](#) function updates the time capacity for all Data Collector tables in the V_MONITOR.QUERY_REQUESTS view. The new setting overrides any previous settings for every Data Collector table in that view.

Examples

The following command configures the Backups component to be retained on disk for 1 day:

```
=> SELECT set_data_collector_time_policy('Backups', '1 day'::interval);
set_data_collector_time_policy
-----
SET
(1 row)
```

This command disables the 1-day restraint for the Backups component:

```
=> SELECT set_data_collector_time_policy('Backups', '-1');
set_data_collector_time_policy
-----
SET
(1 row)
```

This command sets a 30-minute time capacity for all Data Collector tables in a single command:

```
=> SELECT set_data_collector_time_policy('30 minutes'::interval);
set_data_collector_time_policy
-----
SET
(1 row)
```

To view current retention policy settings for each Data Collector table, call the [GET_DATA_COLLECTION_POLICY\(\)](#) function. In the next example, the time restraint is included.

```
=> SELECT get_data_collector_policy('RequestsIssued');
           get_data_collector_policy
-----
2000KB kept in memory, 50000KB kept on disk. 2 years 3 days 15:08 hours kept
on disk.
(1 row)
```

If the time policy setting is disabled, the output of `GET_DATA_COLLECTION_POLICY()` returns "Time based retention disabled."

```
2000KB kept in memory, 50000KB kept on disk. Time based retention disabled.
```

See Also

- [GET_DATA_COLLECTOR_POLICY](#)
- [SET_DATA_COLLECTOR_POLICY](#)
- [DATA_COLLECTOR](#)

Database Designer Functions

Database Designer functions provide programmatic access to Database Designer functionality. For information on required privileges, see [Privileges for Running Database Designer Functions](#) in the Administrator's Guide

Database Designer functions perform the following operations, generally performed in the following order:

1. [Create a design.](#)
2. [Set design properties.](#)
3. [Populate a design.](#)
4. [Create design and deployment scripts.](#)
5. [Get design data.](#)
6. [Clean up.](#)

For detailed information, see [Workflow for Running Database Designer Programmatically.](#)

Create a design

[DESIGNER_CREATE_DESIGN](#) directs Database Designer to create a design.

Set design properties

The following functions let you specify properties of a particular design:

DESIGNER_SET_DESIGN_TYPE	Specifies whether the design is comprehensive or incremental.
DESIGNER_DESIGN_PROJECTION_ENCODINGS	Analyzes encoding in the specified projections and creates a script that implements encoding recommendations.
DESIGNER_SET_DESIGN_KSAFETY	Sets the K-safety value for a comprehensive design.
DESIGNER_SET_	Specifies whether the design optimizes for query or load

OPTIMIZATION_OBJECTIVE	performance.
DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS	Enables inclusion of unsegmented projections in the design.
DESIGNER_SET_ANALYZE_CORRELATIONS_MODE	Determines how the design handles column correlations.

Populate a design

The following functions let you add tables and queries to your Database Designer design:

DESIGNER_ADD_DESIGN_TABLES	Adds the specified tables to a design.
DESIGNER_ADD_DESIGN_QUERY	Adds queries to the design and weights them.
DESIGNER_ADD_DESIGN_QUERIES	
DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS	

Create design and deployment scripts

The following functions populate the Database Designer workspace and create design and deployment scripts. You can also analyze statistics, deploy the design automatically, and drop the workspace after the deployment:

DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY	Populates the design and creates design and deployment scripts.
DESIGNER_WAIT_FOR_DESIGN	Waits for a currently running design to complete.

Reset a design

[DESIGNER_RESET_DESIGN](#) discards all the run-specific information of the previous Database Designer build or deployment of the specified design but retains its configuration.

Get design data

The following functions display information about projections and scripts that the Database Designer created:

DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS	Sends to standard output DDL statements that define design projections.
DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT	Sends to standard output a design's deployment script.

Clean up

The following functions cancel any running Database Designer operation or drop a Database Designer design and all its contents:

DESIGNER_CANCEL_POPULATE_DESIGN	Cancels population or deployment operation for the specified design if it is currently running.
DESIGNER_DROP_DESIGN	Removes the schema associated with the specified design and all its contents.
DESIGNER_DROP_ALL_DESIGNS	Removes all Database Designer-related schemas associated with the current user.

DESIGNER_ADD_DESIGN_QUERIES

Reads and evaluates queries from an input file, and adds the queries that it accepts to the specified design. All accepted queries are assigned a weight of 1 (see [DESIGNER_ADD_DESIGN_QUERIES](#)).

The following requirements apply:

- All queried tables must previously be added to the design with [DESIGNER_ADD_DESIGN_TABLES](#).
- If the [design type](#) is incremental, the Database Designer reads only the first 100 queries in the input file, and ignores all queries beyond that number.

All accepted queries are added to the system table [DESIGN_QUERIES](#).

Behavior Type

Immutable

Syntax

```
DESIGNER_ADD_DESIGN_QUERIES ( 'design-name', 'input-file' [, return-results] )
```

Parameters

<i>design-name</i>	Name of the target design.
<i>input-file</i>	Absolute path to the queries file.
<i>return-results</i>	Boolean, optionally specifies whether to return results of the add operation to standard output. If set to true, Database Designer returns the following results: <ul style="list-style-type: none">• Number of accepted queries• Number of queries referencing non-design tables• Number of unsupported queries• Number of illegal queries

Privileges

Read access to the input file, and one of the following:

- Superuser
- Design creator with all privileges that pertain to the input file queries (see [Privileges Required for Common Database Operations](#)).

Errors

Database Designer returns an error in the following cases:

- The query contains illegal syntax.
- The query references:
 - External or system tables only
 - Local temporary or other non-design tables
- DELETE or UPDATE query has one or more subqueries.
- INSERT query does not include a SELECT clause.
- Database Designer cannot optimize the query.

Examples

The following example adds queries from `vmart_queries.sql` to the `VMART_DESIGN` design. This file contains nine queries. The statement includes a third argument of `true`, so Database Designer returns results of the add operation:

```
=> SELECT DESIGNER_ADD_DESIGN_QUERIES ('VMART_DESIGN', '/tmp/examples/vmart_queries.sql', 'true');
...
DESIGNER_ADD_DESIGN_QUERIES
-----
Number of accepted queries          =9
Number of queries referencing non-design tables =0
Number of unsupported queries       =0
Number of illegal queries           =0
(1 row)
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS

Executes the specified query and evaluates results in the following columns:

- `QUERY_TEXT` (required): Text of potential design queries.
- `QUERY_WEIGHT` (optional): The weight assigned to each query that indicates its importance relative to other queries, a real number >0 and ≤ 1 . Database Designer uses this setting when creating the design to prioritize the query. If `DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS` returns any results that omit this value, Database Designer sets their weight to 1.

After evaluating the queries in `QUERY_TEXT`, `DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS` adds all accepted queries to the design. An unlimited number of queries can be added to the design.

Before you add queries to a design, you must add the queried tables with [DESIGNER_ADD_DESIGN_TABLES](#).

Behavior Type

Immutable

Syntax

```
DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS ( 'design-name', 'query' )
```

Parameters

<i>design-name</i>	Name of the target design.
<i>query</i>	A valid SQL query whose results contain columns named <code>QUERY_TEXT</code> and, optionally, <code>QUERY_WEIGHT</code> .

Privileges

- Superuser
- Design creator with all privileges that pertain to the specified query and all queries that this function returns (see [Privileges Required for Common Database Operations](#)).

Errors

Database Designer returns an error in the following cases:

- The query contains illegal syntax.
- The query references:
 - External or system tables only
 - Local temporary or other non-design tables
- DELETE or UPDATE query has one or more subqueries.

- INSERT query does not include a SELECT clause.
- Database Designer cannot optimize the query.

Example

The following example queries the system table [QUERY_REQUESTS](#) for all long-running queries (> 1 million microseconds) and adds them to the VMART_DESIGN design. The query returns no information on query weights, so all queries are assigned a weight of 1:

```
=> SELECT DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS ( 'VMART_DESIGN',  
  'SELECT request as query_text FROM query_requests where request_duration_ms > 1000000 AND request_  
type =  
  'QUERY' );
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_ADD_DESIGN_QUERY

Reads and parses the specified query, and if accepted, adds it to the design. Before you add queries to a design, you must add the queried tables with [DESIGNER_ADD_DESIGN_TABLES](#).

All accepted queries are added to the system table [DESIGN_QUERIES](#).

Behavior Type

Immutable

Syntax

```
DESIGNER_ADD_DESIGN_QUERY ( 'design-name', 'design-query' [, query-weight] )
```

Parameters

<i>design-name</i>	Name of the target design.
<i>design-query</i>	Executable SQL query.
<i>query-weight</i>	Optionally assigns a weight to each query that indicates its importance

	<p>relative to other queries, a real number >0 and ≤ 1. Database Designer uses this setting to prioritize queries in the design .</p> <p>If you omit this parameter, Database Designer assigns a weight of 1.</p>
--	---

Privileges

One of the following:

- Superuser
- Design creator with all privileges that pertain to the specified query (see [Privileges Required for Common Database Operations](#)).

Errors

Database Designer returns an error in the following cases:

- The query contains illegal syntax.
- The query references:
 - External or system tables only
 - Local temporary or other non-design tables
- DELETE or UPDATE query has one or more subqueries.
- INSERT query does not include a SELECT clause.
- Database Designer cannot optimize the query.

Examples

The following example adds the specified query to the VMART_DESIGN design and assigns that query a weight of 0.5:

```
=> SELECT DESIGNER_ADD_DESIGN_QUERY (  
    'VMART_DESIGN',  
    'SELECT customer_name, customer_type FROM customer_dimension ORDER BY customer_name ASC;', 0.5  
);
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_ADD_DESIGN_TABLES

Adds the specified tables to a design. You must run `DESIGNER_ADD_DESIGN_TABLES` before adding design queries to the design. If no tables are added to the design, Vertica does not accept design queries.

Behavior Type

Immutable

Syntax

```
DESIGNER_ADD_DESIGN_TABLES ( 'design-name', '[ table-spec[,...] ]' [, 'analyze-statistics' ] )
```

Parameters

<i>design-name</i>	Name of the Database Designer design.
<i>table-spec</i> [, ...]	<p>One or more comma-delimited arguments that specify which tables to add to the design, where each <i>table-spec</i> argument can specify tables as follows:</p> <ul style="list-style-type: none">• <i>[schema.]table</i> Add <i>table</i> to the design.• <i>schema.*</i> Add all tables in <i>schema</i>. <p>If set to an empty string, Vertica adds all tables in the database to which the user has access.</p>
<i>analyze-statistics</i>	<p>Boolean that optionally specifies whether to run ANALYZE_STATISTICS after adding the specified tables to the design, by default set to <code>false</code>.</p> <p>Accurate statistics help Database Designer optimize compression and query performance. Updating statistics takes time and resources.</p>

Privileges

One of the following:

- Superuser
- DBDUSER who created the design, has USAGE privilege on the design table schema, and owns the design table.

Examples

The following example adds to design VMART_DESIGN all tables from schemas `online_sales` and `store`, and analyzes statistics for those tables:

```
=> SELECT DESIGNER_ADD_DESIGN_TABLES('VMART_DESIGN', 'online_sales.*', 'store.*','true');
DESIGNER_ADD_DESIGN_TABLES
-----
                          7
(1 row)
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_CANCEL_POPULATE_DESIGN

Cancels population or deployment operation for the specified design if it is currently running. When you cancel a deployment, the Database Designer cancels the projection refresh operation. It does not roll back projections that it already deployed and refreshed.

Behavior Type

Immutable

Syntax

```
DESIGNER_CANCEL_POPULATE_DESIGN ( 'design-name' )
```

Parameters

<i>design-name</i>	Name of the design operation to cancel.
--------------------	---

Privileges

One of the following:

- Superuser
- Design creator

Examples

The following example cancels a currently running design for VMART_DESIGN and then drops the design:

```
=> SELECT DESIGNER_CANCEL_POPULATE_DESIGN ('VMART_DESIGN');  
=> SELECT DESIGNER_DROP_DESIGN ('VMART_DESIGN', 'true');
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_CREATE_DESIGN

Creates a design with the specified name.

Note: Be sure to back up the current design using the function [EXPORT_CATALOG](#) before running the Database Designer functions on an existing schema. You must explicitly back up the existing design when using Database Designer programmatically.

If any of the following [V_MONITOR](#) tables do not already exist from previous designs, DESIGNER_CREATE_DESIGN creates them:

- [DESIGNS](#)
- [DESIGN_TABLES](#)
- [DEPLOYMENT_PROJECTIONS](#)
- [DEPLOYMENT_PROJECTION_STATEMENTS](#)
- [DESIGN_QUERIES](#)

- [OUTPUT_DEPLOYMENT_STATUS](#)
- [OUTPUT_EVENT_HISTORY](#)

Behavior Type

Immutable

Syntax

```
DESIGNER_CREATE_DESIGN ( 'design-name' )
```

Parameters

<i>design-name</i>	Name of the design to create, can contain only alphanumeric and underscore (<code>_</code>) characters. Two users cannot have designs with the same name at the same time.
--------------------	---

Privileges

One of the following:

- Superuser
- DBDUSER

Examples

The following example creates the design VMART_DESIGN:

```
=> SELECT DESIGNER_CREATE_DESIGN('VMART_DESIGN');  
DESIGNER_CREATE_DESIGN  
-----  
0  
(1 row)
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_DESIGN_PROJECTION_ENCODINGS

Analyzes encoding in the specified projections, creates a script to implement encoding recommendations, and optionally deploys the recommendations.

Behavior Type

Immutable

Syntax

```
DESIGNER_DESIGN_PROJECTION_ENCODINGS ( '[ proj-spec[,... ] ]', '[destination]' [, 'deploy'] [, 'reanalyze-encodings' ] )
```

Parameters

<code><i>proj-spec</i>[, ...]</code>	<p>One or more comma-delimited projections to add to the design. Each projection can be specified in one of the following ways:</p> <ul style="list-style-type: none">• <code>[[<i>schema</i>.]<i>table</i>.]<i>projection</i></code> Specifies to analyze <i>projection</i>.• <code><i>schema</i>.*</code> Specifies to analyze all projections in the named schema.• <code>[[<i>schema</i>.]<i>table</i></code> Specifies to analyze all projections of the named table. <p>If set to an empty string, Vertica analyzes all projections in the database to which the user has access.</p> <p>For example, the following statement specifies to analyze all projections in schema <code>private</code>, and send the results to the file <code>encodings.sql</code>:</p> <pre>=> SELECT DESIGNER_DESIGN_PROJECTION_ENCODINGS ('mydb.private.*', 'encodings.sql');</pre>
<code><i>destination</i></code>	<p>Specifies where to send output, one of the following:</p> <ul style="list-style-type: none">• An empty string (' ') writes the script to standard output.

	<ul style="list-style-type: none"> The path and name of a SQL output file. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.
<i>deploy</i>	Boolean that optionally specifies whether to deploy encoding changes, by default set to <code>false</code> .
<i>reanalyze-encodings</i>	<p>Boolean that optionally specifies whether <code>DESIGNER_DESIGN_PROJECTION_ENCODINGS</code> analyzes encodings in a projection where all columns are already encoded:</p> <ul style="list-style-type: none"> <code>false</code> (default): Analyzes no columns and generates no recommendations if all columns are encoded. <code>true</code>: Ignores existing encodings and generates recommendations.

Privileges

The following requirements pertain to the target projections:

- OWNER of all projections to analyze
- USAGE privilege on the schema for the specified projections

Examples

The following example requests that Database Designer analyze encodings of all projections in the schema `online_sales`, and save output to the file `encodings.sql`. The last parameter is omitted, so Database Designer does not execute the script:

```
=> SELECT DESIGNER_DESIGN_PROJECTION_ENCODINGS ('online_sales.*', 'encodings.sql');
DESIGNER_DESIGN_PROJECTION_ENCODINGS
-----
(1 row)
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_DROP_ALL_DESIGNS

Removes all Database Designer-related schemas associated with the current user. Use this function to remove database objects after one or more Database Designer sessions complete execution.

Behavior Type

Immutable

Syntax

```
DESIGNER_DROP_ALL_DESIGNS()
```

Parameters

None.

Privileges

One of the following:

- Superuser : Drops all designs.
- Design creator: Drops all designs created by this user.

Example

The following example removes all schema and their contents associated with the current user. `DESIGNER_DROP_ALL_DESIGNS` returns the number of designs dropped:

```
=> SELECT DESIGNER_DROP_ALL_DESIGNS();  
DESIGNER_DROP_ALL_DESIGNS  
-----  
2  
(1 row)
```

See Also

- [DESIGNER_CANCEL_POPULATE_DESIGN](#)
- [DESIGNER_DROP_DESIGN](#)

DESIGNER_DROP_DESIGN

Removes the schema associated with the specified design and all its contents. Use `DESIGNER_DROP_DESIGN` after a Database Designer design or deployment completes successfully. You must also use it to drop a design before creating another one under the same name.

To drop all designs that you created, use [DESIGNER_DROP_ALL_DESIGNS](#).

Behavior Type

Immutable

Syntax

```
DESIGNER_DROP_DESIGN ( 'design-name' [, force-drop ] )
```

Parameters

<i>design-name</i>	Name of the design to drop.
<i>force-drop</i>	Boolean that overrides any dependencies that otherwise prevent Vertica from executing this function—for example, the design is in use or is currently being deployed. If you omit this parameter, Vertica sets it to false.

Privileges

One of the following:

- Superuser
- Design creator

Example

The following example deletes the Database Designer design VMART_DESIGN and all its contents:

```
=> SELECT DESIGNER_DROP_DESIGN ('VMART_DESIGN');
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS

Displays the DDL statements that define the design projections to standard output.

Behavior Type

Immutable

Syntax

```
DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS ( 'design-name' )
```

Parameters

<i>design-name</i>	Name of the target design.
--------------------	----------------------------

Privileges

One of the following:

- Superuser
- DBDUSER role

Examples

The following example returns the design projection DDL statements for vmart_design:

```
=> SELECT DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS('vmart_design');
CREATE PROJECTION customer_dimension_DBD_1_rep_VMART_DESIGN /*+createtype(D)*/
(
  customer_key ENCODING DELTAVAL,
  customer_type ENCODING AUTO,
  customer_name ENCODING AUTO,
  customer_gender ENCODING REL,
  title ENCODING AUTO,
  household_id ENCODING DELTAVAL,
  customer_address ENCODING AUTO,
  customer_city ENCODING AUTO,
  customer_state ENCODING AUTO,
  customer_region ENCODING AUTO,
  marital_status ENCODING AUTO,
  customer_age ENCODING DELTAVAL,
  number_of_children ENCODING BLOCKDICT_COMP,
  annual_income ENCODING DELTARANGE_COMP,
  occupation ENCODING AUTO,
  largest_bill_amount ENCODING DELTAVAL,
  store_membership_card ENCODING BLOCKDICT_COMP,
  customer_since ENCODING DELTAVAL,
  deal_stage ENCODING AUTO,
  deal_size ENCODING DELTARANGE_COMP,
  last_deal_update ENCODING DELTARANGE_COMP
)
AS
SELECT customer_key,
       customer_type,
       customer_name,
       customer_gender,
       title,
       household_id,
       customer_address,
       customer_city,
       customer_state,
       customer_region,
       marital_status,
       customer_age,
       number_of_children,
       annual_income,
       occupation,
       largest_bill_amount,
       store_membership_card,
       customer_since,
       deal_stage,
       deal_size,
       last_deal_update
FROM public.customer_dimension
ORDER BY customer_gender,
        annual_income
UNSEGMENTED ALL NODES;
CREATE PROJECTION product_dimension_DBD_2_rep_VMART_DESIGN /*+createtype(D)*/
(
  ...
```

See Also

[DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT](#)

DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT

Displays the deployment script for the specified design to standard output. If the design is already deployed, Vertica ignores this function.

To output only the CREATE PROJECTION commands in a design script, use [DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS](#).

Behavior Type

Immutable

Syntax

```
DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT ( 'design-name' )
```

Parameters

<i>design-name</i>	Name of the target design.
--------------------	----------------------------

Privileges

One of the following:

- Superuser
- Design creator

Examples

The following example displays the deployment script for VMART_DESIGN:

```
=> SELECT DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT('VMART_DESIGN');  
CREATE PROJECTION customer_dimension_DBD_1_rep_VMART_DESIGN /*+createtype(D)*/  
...  
CREATE PROJECTION product_dimension_DBD_2_rep_VMART_DESIGN /*+createtype(D)*/  
...  
select refresh('public.customer_dimension,  
              public.product_dimension,  
              public.promotion.dimension,  
              public.date_dimension');  
select make_ahm_now();
```

```
DROP PROJECTION public.customer_dimension_super CASCADE;  
DROP PROJECTION public.product_dimension_super CASCADE;  
...
```

See Also

[DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS](#)

DESIGNER_RESET_DESIGN

Discards all run-specific information of the previous Database Designer build or deployment of the specified design but keeps its configuration. You can make changes to the design as needed, for example, by changing parameters or adding additional tables and/or queries, before running the design again.

Behavior Type

Immutable

Syntax

```
DESIGNER_RESET_DESIGN ( 'design-name' )
```

Parameters

<i>design-name</i>	Name of the design to reset.
--------------------	------------------------------

Privileges

- Superuser
- Design creator

Example

The following example resets the Database Designer design VMART_DESIGN:

```
=> SELECT DESIGNER_RESET_DESIGN ('VMART_DESIGN');
```

DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY

Populates the design and creates the design and deployment scripts. `DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY` can also analyze statistics, deploy the design, and drop the workspace after the deployment.

Caution: `DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY` does not create a backup copy of the current design before deploying the new design. Before running this function, back up the existing schema design with [EXPORT_CATALOG](#).

Behavior Type

Immutable

Syntax

```
DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY (  
  'design-name',  
  'output-design-file',  
  'output-deployment-file',  
  [ 'analyze-statistics', ]  
  [ 'deploy', ]  
  [ 'drop-design-workspace', ]  
  [ 'continue-after-error', ]  
)
```

Parameters

<i>design-name</i>	Name of the design to populate and deploy.
<i>output-design-file</i>	Specifies where to save the file with DDL statements to create design projections, where <i>output-design-file</i> is an absolute path to the node where the session is connected.
<i>output-deployment-file</i>	Specifies where to save the file that contains the deployment script, where <i>output-deployment-file</i> is an absolute path to the node where the session is connected.
<i>analyze-statistics</i>	Specifies whether to collect or refresh statistics for the tables before populating the design. If set to true, Vertica

	<p>Invokes ANALYZE_STATISTICS. Accurate statistics help Database Designer optimize compression and query performance. However, updating statistics requires time and resources.</p> <p>Default: false</p>
<i>deploy</i>	<p>Specifies whether to deploy the Database Designer design using the deployment script created by this function.</p> <p>Default: true</p>
<i>drop-design-workspace</i>	<p>Specifies whether to drop the design workspace after the design is deployed.</p> <p>Default: true</p>
<i>continue-after-error</i>	<p>Specifies whether DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY continues to run after an error occurs. By default, an error causes this function to terminate.</p> <p>Default: false</p>

Privileges

- Superuser
- Design creator with WRITE privileges on storage locations of design and deployment scripts; otherwise DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY cannot save the design and deployment scripts.

Requirements

Before calling this function, you must:

- Create a design, a logical schema with tables.
- Associate tables with the design.
- Load queries to the design.
- Set design properties (K-safety level, mode, and policy).

Examples

The following example creates projections for and deploys the VMART_DESIGN design, and analyzes statistics about the design tables.

```
=> SELECT DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY (  
    'VMART_DESIGN',  
    '/tmp/examples/vmart_design_files/vmart_design_DDL',  
    '/tmp/examples/vmart_design_files/vmart_design_deployment_scripts',  
    'true',  
    'false',  
    'false',  
    'false'  
);
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_SET_ANALYZE_CORRELATIONS_MODE

Specifies how Database Designer handles column correlations in a design. A design's mode determines whether Database Designer analyzes or re-analyzes existing column correlations and considers them in the design that it creates.

The following recommendations apply:

- You typically need to analyze column correlations only once.
- Analyze correlations when the table row count is at least [DBDCorrelationSampleRowCount](#)—by default, 4000.

Important: Database Designer analyzes column correlations for a design only if you enable analysis with this function.

Behavior Type

Immutable

Syntax

```
DESIGNER_SET_ANALYZE_CORRELATIONS_MODE ( 'design-name', mode )
```

Parameters

<i>design-name</i>	Name of the design that specifies how Database Designer handles correlated columns.								
<i>mode</i>	<p>Specifies how to handle correlations in the design tables, one of the following integer settings:</p> <table border="1"> <tr> <td>0</td> <td>Ignore column correlations in the design tables.</td> </tr> <tr> <td>1</td> <td>Consider existing correlations in tables when creating the design. If no existing correlations exist, Database Designer sets none in the design.</td> </tr> <tr> <td>2</td> <td>Analyze column correlations on tables where correlation analysis was not previously performed. When creating the design, consider all column correlations, new and existing.</td> </tr> <tr> <td>3</td> <td>Analyze all tables for column correlations and consider them when creating the design. If correlations already exist for a table, ignore them and re-analyze the table.</td> </tr> </table> <p>Setting the correlation analysis mode does not affect whether Database Designer analyzes statistics when creating a design.</p>	0	Ignore column correlations in the design tables.	1	Consider existing correlations in tables when creating the design. If no existing correlations exist, Database Designer sets none in the design.	2	Analyze column correlations on tables where correlation analysis was not previously performed. When creating the design, consider all column correlations, new and existing.	3	Analyze all tables for column correlations and consider them when creating the design. If correlations already exist for a table, ignore them and re-analyze the table.
0	Ignore column correlations in the design tables.								
1	Consider existing correlations in tables when creating the design. If no existing correlations exist, Database Designer sets none in the design.								
2	Analyze column correlations on tables where correlation analysis was not previously performed. When creating the design, consider all column correlations, new and existing.								
3	Analyze all tables for column correlations and consider them when creating the design. If correlations already exist for a table, ignore them and re-analyze the table.								

Privileges

One of the following:

- Superuser
- DBDUSER role with USAGE privilege on the design schema.

Example

The following example specifies that Database Designer analyze all tables for correlated columns and consider them when creating a design:

```
=> SELECT DESIGNER_SET_ANALYZE_CORRELATIONS_MODE ('VMART_DESIGN', 3);
DESIGNER_SET_ANALYZE_CORRELATIONS_MODE
```

3

(1 row)

See Also

- [ANALYZE_CORRELATIONS](#)
- [Running Database Designer Programmatically](#)

DESIGNER_SET_DESIGN_KSAFETY

Sets [K-safety](#) for a comprehensive design and stores the K-safety value in the [DESIGNS](#) table. Database Designer ignores this function for incremental designs.

Behavior Type

Immutable

Syntax

```
DESIGNER_SET_DESIGN_KSAFETY ( 'design-name' [, k-Level ]  
)
```

Parameters

<i>design-name</i>	Name of the design for which you want to set the K-safety value, type VARCHAR.
<i>k-Level</i>	<p>An integer between 0 and 2 that specifies the level of K-safety for the target design. This value must be compatible with the number of nodes in the database cluster:</p> <ul style="list-style-type: none">• <i>k-Level</i> = 0: ≥ 1 nodes• <i>k-Level</i> = 1: ≥ 3 nodes• <i>k-Level</i> = 2: ≥ 5 nodes <p>If you omit this parameter, Vertica sets K-safety for this design to 0 or 1, according to the number of nodes: 1 if the cluster contains ≥ 3 nodes, otherwise 0.</p>

	<p>If you are a DBADMIN user and <i>k-Level</i> differs from system K-safety, Vertica changes system K-safety as follows:</p> <ul style="list-style-type: none">• If <i>k-Level</i> is less than system K-safety, Vertica changes system K-safety to the lower level after the design is deployed.• If <i>k-Level</i> is greater than system K-safety and is valid for the database cluster, Vertica creates the required number of buddy projections for the tables in this design. If the design applies to all database tables, or all tables in the database have the required number of buddy projections, Database Designer changes system K-safety to <i>k-Level</i>. <p>If the design excludes some database tables and the number of their buddy projections is less than <i>k-Level</i>, Database Designer leaves system K-safety unchanged. Instead, it returns a warning and indicates which tables need new buddy projections in order to adjust system K-safety.</p> <p>If you are a DBDUSER, Vertica ignores this parameter.</p>
--	--

Privileges

One of the following:

- Superuser
- Design creator

Examples

The following example set K-safety for the VMART_DESIGN design to 1:

```
=> SELECT DESIGNER_SET_DESIGN_KSAFETY('VMART_DESIGN', 1);
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_SET_DESIGN_TYPE

Specifies whether Database Designer should create a comprehensive or incremental design. `DESIGNER_SET_DESIGN_TYPE` stores the design mode in the [DESIGNS](#) table.

If you do not explicitly set a design mode with this function, Database Designer creates a comprehensive design.

Behavior Type

Immutable

Syntax

```
DESIGNER_SET_DESIGN_TYPE ( 'design-name', 'mode' )
```

Parameters

<i>design-name</i>	Name of the target design.
<i>mode</i>	<p>Name of the mode that Database Designer should use when designing the database, one of the following:</p> <ul style="list-style-type: none">• COMPREHENSIVE: Creates an initial or replacement design for all tables in the specified schemas. You typically create a comprehensive design for a new database.• INCREMENTAL: Modifies an existing design with additional projection that are optimized for new or modified queries. <p>For more information, see Design Types in the Administrator's Guide.</p>

Privileges

One of the following:

- Superuser
- Design creator

Notes

Incremental designs always inherit the K-safety value of the database.

Examples

The following examples show the two design mode options for the VMART_DESIGN design:

```
=> SELECT DESIGNER_SET_DESIGN_TYPE(  
    'VMART_DESIGN',  
    'COMPREHENSIVE');  
DESIGNER_SET_DESIGN_TYPE  
-----  
                                0  
(1 row)  
=> SELECT DESIGNER_SET_DESIGN_TYPE(  
    'VMART_DESIGN',  
    'INCREMENTAL');  
DESIGNER_SET_DESIGN_TYPE  
-----  
                                0  
(1 row)
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_SET_OPTIMIZATION_OBJECTIVE

Valid only for comprehensive database designs, specifies the optimization objective Database Designer uses. Database Designer ignores this function for incremental designs.

DESIGNER_SET_OPTIMIZATION_OBJECTIVE stores the optimization objective in the [DESIGNS](#) table.

Behavior Type

Immutable

Syntax

```
DESIGNER_SET_OPTIMIZATION_OBJECTIVE ( 'design-name', 'policy' )
```

Parameters

<i>design-name</i>	Name of the target design.
<i>policy</i>	Specifies the design's optimization policy, one of the following: <ul style="list-style-type: none">• QUERY: Optimize for query performance. This can result in a larger database storage footprint because additional projections might be created.• LOAD: Optimize for load performance so database size is minimized. This can result in slower query performance.• BALANCED: Balance the design between query performance and database size.

Privileges

One of the following:

- Superuser
- Design creator

Examples

The following example sets the optimization objective option for the VMART_DESIGN design: to QUERY:

```
=> SELECT DESIGNER_SET_OPTIMIZATION_OBJECTIVE( 'VMART_DESIGN', 'QUERY');
DESIGNER_SET_OPTIMIZATION_OBJECTIVE
-----
                                0
(1 row)
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS

Specifies whether a design can include [unsegmented projections](#). Vertica ignores this function on a one-node cluster, where all projections must be unsegmented.

Behavior Type

Immutable

Syntax

```
DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS ( 'design-name', unsegmented )
```

Parameters

<i>design-name</i>	Name of the target design.
<i>unsegmented</i>	Boolean that specifies whether Database Designer can propose unsegmented projections for tables in this design. When you create a design, the <code>propose_unsegmented_projections</code> value in system table DESIGNS for this design is set to <code>true</code> . If <code>DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS</code> sets this value to <code>false</code> , Database Designer only proposes segmented projections for the design.

Privileges

One of the following:

- Superuser
- Design creator

Example

The following example specifies that Database Designer can propose only segmented projections for tables in the design `VMART_DESIGN`:

```
=> SELECT DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS('VMART_DESIGN', false);
```

See Also

[Running Database Designer Programmatically](#)

DESIGNER_WAIT_FOR_DESIGN

Waits for completion of operations that are populating and deploying the design. Ctrl+C cancels this operation and returns control to the user.

Behavior Type

Immutable

Syntax

```
DESIGNER_WAIT_FOR_DESIGN ( 'design-name' )
```

Parameters

<i>design-name</i>	Name of the running design.
--------------------	-----------------------------

Privileges

Privileges

One of the following:

- Superuser
- DBDUSER role with USAGE privilege on the design schema.

Examples

The following example requests to wait for the currently running design of VMART_DESIGN to complete:

```
=> SELECT DESIGNER_WAIT_FOR_DESIGN ('VMART_DESIGN');
```

See Also

- [DESIGNER_CANCEL_POPULATE_DESIGN](#)
- [DESIGNER_DROP_ALL_DESIGNS](#)
- [DESIGNER_DROP_DESIGN](#)

Database Management Functions

This section contains the database management functions specific to Vertica.

CLEAR_RESOURCE_REJECTIONS

Clears the content of the [RESOURCE_REJECTIONS](#) and [DISK_RESOURCE_REJECTIONS](#) system tables. Normally, these tables are only cleared during a node restart. This function lets you clear the tables whenever you need. For example, you might want to clear the system tables after you resolved a disk space issue that was causing disk resource rejections.

Syntax

```
CLEAR_RESOURCE_REJECTIONS();
```

Privileges

Superuser

Example

The following command clears the content of the `RESOURCE_REJECTIONS` and `DISK_RESOURCE_REJECTIONS` system tables:

```
=> SELECT clear_resource_rejections();
clear_resource_rejections
-----
OK
(1 row)
```

See Also

- [DISK_RESOURCE_REJECTIONS](#)
- [RESOURCE_REJECTIONS](#)

CURRENT_SCHEMA

Returns the name of the current schema.

Behavior Type

Stable

Syntax

```
CURRENT_SCHEMA()
```

Note: You can call this function without parentheses.

Privileges

None

Examples

The following command returns the name of the current schema:

```
=> SELECT CURRENT_SCHEMA();
current_schema
-----
public
(1 row)
```

The following command returns the same results without the parentheses:

```
=> SELECT CURRENT_SCHEMA;
current_schema
-----
public
(1 row)
```

The following command shows the current schema, listed after the [current user](#), in the search path:

```
=> SHOW SEARCH_PATH;
name | setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

See Also

- [SET SEARCH_PATH](#)

DUMP_LOCKTABLE

Returns information about deadlocked clients and the resources they are waiting for.

Syntax

```
DUMP_LOCKTABLE()
```

Privileges

None

Notes

Use DUMP_LOCKTABLE if Vertica becomes unresponsive:

1. Open an additional vsql connection.
2. Execute the query:

```
=> SELECT DUMP_LOCKTABLE();
```

The output is written to vsql. See [Monitoring the Log Files](#).

You can also see who is connected using the following command:

```
=> SELECT * FROM SESSIONS;
```

Close all sessions using the following command:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

Close a single session using the following command:

```
=> SELECT CLOSE_SESSION('session_id');
```

You get the session_id value from the [V_MONITOR.SESSIONS](#) system table.

See Also

- [CLOSE_ALL_SESSIONS](#)
- [CLOSE_SESSION](#)
- [LOCKS](#)
- [SESSIONS](#)

DUMP_PARTITION_KEYS

Dumps the partition keys of all projections in the system.

Syntax

```
DUMP_PARTITION_KEYS( )
```

Note: The ROS objects of partitioned tables without partition keys are ignored by the tuple mover and are not merged during automatic tuple mover operations.

Privileges

User must have select privileges on the table or usage privileges on the schema.

Example

```
=> SELECT DUMP_PARTITION_KEYS( );
Partition keys on node v_vmart_node0001
Projection 'states_b0'
Storage [ROS container]
  No of partition keys: 1
  Partition keys: NH
Storage [ROS container]
  No of partition keys: 1
  Partition keys: MA
Projection 'states_b1'
Storage [ROS container]
  No of partition keys: 1
  Partition keys: VT
Storage [ROS container]
  No of partition keys: 1
  Partition keys: ME
Storage [ROS container]
  No of partition keys: 1
```

Partition keys: CT

See Also

- [DO_TM_TASK](#)
- [DROP_PARTITION](#)
- [DUMP_PROJECTION_PARTITION_KEYS](#)
- [DUMP_TABLE_PARTITION_KEYS](#)
- [PARTITION_PROJECTION](#)
- [PARTITION_TABLE](#)
- [PARTITIONS](#)
- [Using Table Partitions](#)

HAS_ROLE

Indicates, with a Boolean value, whether a role has been assigned to a user. This function is useful for letting you check your own role membership.

Behavior Type

Stable

Syntax 1

```
HAS_ROLE( [ 'user_name' ,] 'role_name' );
```

Syntax 2

```
HAS_ROLE( 'role_name' );
```

Parameters

<i>user_name</i>	[Optional] The name of a user to look up. Currently, only a superuser can supply the <code>user_name</code> argument.
------------------	---

<code>role_name</code>	The name of the role you want to verify has been granted.
------------------------	---

Privileges

Users can check their own role membership by calling `HAS_ROLE('role_name')`, but only a superuser can look up other users' memberships using the optional `user_name` parameter.

Notes

You can query `V_CATALOG` system tables [ROLES](#), [GRANTS](#), and [USERS](#) to show any directly-assigned roles; however, these tables do not indicate whether a role is available to a user when roles may be available through other roles (indirectly).

Examples

User Bob wants to see if he has been granted the commentor role:

```
=> SELECT HAS_ROLE('commentor');
```

Output `t` for true indicates that Bob has been assigned the commentor role:

```
HAS_ROLE
-----
t
(1 row)
```

In the following function call, a superuser checks if the logadmin role has been granted to user Bob:

```
=> SELECT HAS_ROLE('Bob', 'logadmin');
HAS_ROLE
-----
t
(1 row)
```

To view the names of all roles users can access, along with any roles that have been assigned to those roles, query the [V_CATALOG.ROLES](#) system table. An asterisk in the output means role granted WITH ADMIN OPTION.

```
=> SELECT * FROM roles;
role_id | name | assigned_roles
-----+-----+-----
45035996273704964 | public |
45035996273704966 | dbduser |
45035996273704968 | dbadmin | dbduser*
45035996273704972 | pseudosuperuser | dbadmin*
```

```
45035996273704974 | logreader      |  
45035996273704976 | logwriter      |  
45035996273704978 | logadmin       | logreader,  
                                     logwriter  
(7 rows)
```

See Also

- [GRANTS](#)
- [ROLES](#)
- [USERS](#)
- [Managing Users and Privileges](#)
- [Viewing a User's Role](#)

KERBEROS_CONFIG_CHECK

Tests the Kerberos configuration of a Vertica cluster. The function performs the following tests, in order:

- Are Kerberos services available?
- Does a keytab file exist and are the Kerberos configuration parameters set in the database?
- Can Vertica read and invoke kinit with the keys?

If any test fails, the function returns a descriptive error message.

Syntax

```
KERBEROS_CONFIG_CHECK( )
```

Parameters

This function has no parameters.

Privileges

This function does not require privileges.

Examples

The following example shows the results when the Kerberos configuration is valid.

```
=> SELECT KERBEROS_CONFIG_CHECK();
ok: kinit exists
ok: klist exists
ok: krb5 exists at [/etc/krb5.conf]
ok: Vertica Keytab file is set to [/scratch_b/qa/vdb.keytab]
ok: Vertica Keytab file exists at [/scratch_b/qa/vdb.keytab]
Kerberos configuration parameters set in the database
KerberosServiceName : [vdb]
KerberosHostname : []
KerberosRealm : [EXAMPLE.COM]
KerberosKeytabFile : [/scratch_b/qa/vdb.keytab]
Vertica Principal: [vdb/engvmqa24.example.com@EXAMPLE.COM]
ok: Can read Vertica keys
ok: Can get tickets for vertica principal
ok: vertica can kinit

(1 row)
```

The following example shows an error report.

```
=> SELECT KERBEROS_CONFIG_CHECK();
WARNING 2807: Could not access file "/etc/krb5.keytab": No such file or directory
ok: kinit exists
ok: klist exists
ok: krb5 exists at [/etc/krb5.conf]
FAILED: Vertica Keytab file is not set
FAILED: Could not find Vertica Keytab file at
Kerberos configuration parameters set in the database
    KerberosServiceName : [vertica]
    KerberosHostname : []
    KerberosRealm : []
    KerberosKeytabFile : []
Vertica Principal: []
FAILED: Command to read Vertica keys did not succeed
FAILED: Command to kinit Vertica keys did not succeed
FAILED: Vertica not 'kinit'ing if krb5 cannot kinit

(1 row)
```

SET_CONFIG_PARAMETER

Specifies the value of a configuration parameter at the database level, or for a specific node.

Important: Vertica encourages use of `ALTER NODE`, `ALTER DATABASE`, and `ALTER SESSION` statements to set and clear configuration parameters. See [Managing Configuration Parameters: VSQL](#) for more information.

Caution: . Vertica is designed to operate with minimal configuration changes, so use this capability sparingly. Carefully follow any documented guidelines for the parameter you wish to configure.

Syntax

```
SET_CONFIG_PARAMETER( 'parameter-name', value, ['node-name'] )
```

Parameters

<i>parameter-name</i>	The parameter value to set. See Configuration Parameters in the Administrator's Guide for a list of supported parameters, their purposes, and usage examples.
<i>value</i>	The value to set for <i>parameter-name</i> . Syntax for this argument varies depending upon the parameter and its expected data type. For strings, enclose the argument in single quotes; integer arguments can be unquoted. If <code>value</code> is specified as <code>NULL</code> , the parameter is cleared.
<i>node-name</i>	The name of the node whose parameter value you wish to set. If you omit this parameter or set it to <code>NULL</code> , the parameter is set at the database level. If a parameter is explicitly set for a node, the node setting supersedes the database-level setting.

Note: Some parameters require restart for the value to take effect.

Privileges

Superuser

Examples

The following examples show how to use `SET_CONFIG_PARAMETER` in various situations.

Set a Configuration Parameter at Database Level

Set the `AnalyzeRowCountInterval` parameter to 3600 at the database level:

```
=> SELECT SET_CONFIG_PARAMETER ('AnalyzeRowCountInterval',3600);
```

Find Details on All Configuration Parameters

Find all configuration parameters and information about them, including their current and default values:

```
=> SELECT * FROM CONFIGURATION_PARAMETERS;
```

See Also

[Managing Configuration Parameters: VSQL](#)

[CONFIGURATION_PARAMETERS](#)

SHUTDOWN

Forces a database to shut down, even if there are users connected.

Syntax

```
SHUTDOWN ( [ 'false' | 'true' ] )
```

Parameters

<i>false</i>	[Default] Returns a message if users are connected. Has the same effect as supplying no parameters.
<i>true</i>	Performs a moveout operation and forces the database to shut down, disallowing further connections.

Privileges

Superuser

Notes

- Quotes around the `true` or `false` arguments are optional.
- Issuing the shutdown command without arguments or with the default (`false`) argument returns a message if users are connected, and the shutdown fails. If no users are connected, the database performs a moveout operation and shuts down.

- Issuing the `SHUTDOWN('true')` command forces the database to shut down whether users are connected or not.
- You can check the status of the shutdown operation in the `vertica.log` file:

```
2010-03-09 16:51:52.625 unknown:0x7fc6d6d2e700  
[Init] <INFO> Shutdown complete. Exiting.
```

- As an alternative to `SHUTDOWN()`, you can also temporarily set `MaxClientSessions` to 0 and then use `CLOSE_ALL_SESSIONS()`. New client connections cannot connect unless they connect using the `dbadmin` account. See [CLOSE_ALL_SESSIONS](#) for details.

Examples

The following command attempts to shut down the database. Because users are connected, the command fails:

```
=> SELECT SHUTDOWN('false');  
NOTICE: Cannot shut down while users are connected  
        SHUTDOWN  
-----  
Shutdown: aborting shutdown  
(1 row)
```

`SHUTDOWN()` and `SHUTDOWN('false')` perform the same operation:

```
=> SELECT SHUTDOWN();  
NOTICE: Cannot shut down while users are connected  
        SHUTDOWN  
-----  
Shutdown: aborting shutdown  
(1 row)
```

Using the `'true'` parameter forces the database to shut down, even though clients might be connected:

```
=> SELECT SHUTDOWN('true');  
        SHUTDOWN  
-----  
Shutdown: moveout complete  
(1 row)
```

See Also

- [SESSIONS](#)

Directed Queries Functions

The following meta-functions let you batch export query plans as directed queries from one Vertica database, and import those directed queries to another database.

EXPORT_DIRECTED_QUERIES

Generates SQL for creating directed queries from a set of input queries, and writes the SQL to the specified file or to standard output.

Syntax

```
EXPORT_DIRECTED_QUERIES('input-file', '[output-file]')
```

Parameters

<i>input-file</i>	A SQL file that contains one or more input queries. See Input Format below for details on format requirements.
<i>output-file</i>	Specifies where to write the generated SQL for creating directed queries. If the file name already exists, EXPORT_DIRECTED_QUERIES returns with an error. If you supply an empty string, Vertica writes the SQL to standard output. See Output Format below for details.

Privileges

Superuser

Input Format

The input file that you supply to EXPORT_DIRECTED_QUERIES contains one or more input queries. For each input query, you can optionally specify two fields that are used in the generated directed query:

- `DirQueryName` provides the directed query's unique identifier, a string that conforms to conventions described in [Identifiers](#).
- `DirQueryComment` specifies a quote-delimited string, up to 128 characters.

You format each input query as follows:

```
--DirQueryName=query-name  
--DirQueryComment='comment'  
input-query
```

Output Format

EXPORT_DIRECTED_QUERIES generates SQL for creating directed queries, and writes the SQL to the specified file or to standard output. In both cases, output conforms to the following format:

```
/* Query: directed-query-name */  
/* Comment: directed-query-comment */  
SAVE QUERY input-query;  
CREATE DIRECTED QUERY CUSTOM 'directed-query-name'  
COMMENT 'directed-query-comment'  
OPTVER 'vertica-release-num'  
PSDATE 'timestamp'  
annotated-query
```

Error Handling

If any errors or warnings occur during EXPORT_DIRECTED_QUERIES execution, it returns with a message like this one:

```
1 queries successfully exported.  
1 warning message was generated.  
Queries exported to /home/dbadmin/outputQueries.  
See error report, /home/dbadmin/outputQueries.err for details.
```

EXPORT_DIRECTED_QUERIES writes all errors and warnings to a file that it creates on the same path as the output file, and uses the output file's base name.

For example:

```
-----  
WARNING: Name field not supplied. Using auto-generated name: 'Autoname:2016-04-25 15:03:32.115317.0'  
Input Query: SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name,  
employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city =  
'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;  
END WARNING
```

Examples

See [Batch Query Plan Export](#) in the Administrator's Guide.

See Also

- [Batch Query Plan Export](#)
- [IMPORT_DIRECTED_QUERIES](#)

IMPORT_DIRECTED_QUERIES

Imports to the database catalog directed queries from a SQL file that was generated by [EXPORT_DIRECTED_QUERIES](#). If no directed queries are specified, Vertica lists all directed queries in the SQL file.

Syntax

```
IMPORT_DIRECTED_QUERIES( 'export-file'[, 'directed-query-name[,...]' ] )
```

Parameters

<i>export-file</i>	A SQL file generated by EXPORT_DIRECTED_QUERIES. When you run this file, Vertica creates the specified directed queries in the current database catalog.
<i>directed-query-name</i>	The name of a directed query that is defined in <i>export-file</i> . You can specify multiple comma-delimited directed query names. If you omit this parameter, Vertica lists the names of all directed queries in <i>export-file</i> .

Privileges

Superuser

See Also

- [Batch Query Plan Export](#)
- [EXPORT_DIRECTED_QUERIES](#)

Epoch Management Functions

This section contains the epoch management functions specific to Vertica.

ADVANCE_EPOCH

Manually closes the current epoch and begins a new epoch.

Syntax

```
ADVANCE_EPOCH ( [ integer ] )
```

Parameters

<i>integer</i>	Specifies the number of epochs to advance.
----------------	--

Privileges

Superuser

Notes

This function is primarily maintained for backward compatibility with earlier versions of Vertica.

Example

The following command increments the epoch number by 1:

```
=> SELECT ADVANCE_EPOCH(1);
```

GET_AHM_EPOCH

Returns the number of the epoch in which the Ancient History Mark is located. Data deleted up to and including the AHM epoch can be purged from physical storage.

Syntax

GET_AHM_EPOCH()

Note: The AHM epoch is 0 (zero) by default (purge is disabled).

Privileges

None

Examples

```
=> SELECT GET_AHM_EPOCH();
      GET_AHM_EPOCH
-----
Current AHM epoch: 0
(1 row)
```

GET_AHM_TIME

Returns a `TIMESTAMP` value representing the Ancient History Mark. Data deleted up to and including the AHM epoch can be purged from physical storage.

Syntax

GET_AHM_TIME()

Privileges

None

Examples

```
=> SELECT GET_AHM_TIME();
      GET_AHM_TIME
-----
Current AHM Time: 2010-05-13 12:48:10.532332-04
(1 row)
```

GET_CURRENT_EPOCH

The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written.

Returns the number of the current epoch.

Syntax

```
GET_CURRENT_EPOCH()
```

Privileges

None

Examples

```
=> SELECT GET_CURRENT_EPOCH();
   GET_CURRENT_EPOCH
-----
                   683
(1 row)
```

GET_LAST_GOOD_EPOCH

Returns the last good epoch number. If no database has no projections, the function returns an error.

Syntax

```
GET_LAST_GOOD_EPOCH()
```

Privileges

None

Examples

```
=> SELECT GET_LAST_GOOD_EPOCH();
       GET_LAST_GOOD_EPOCH
-----
                682
(1 row)
```

MAKE_AHM_NOW

Sets the Ancient History Mark (AHM) to the greatest allowable value. This lets you purge all deleted data.

Caution: After running this function, you cannot query historical data that precedes the current epoch. Only database administrators should use this function.

MAKE_AHM_NOW performs the following operations:

- Advances the epoch.
- Performs a [moveout](#) operation on all projections.
- Sets the AHM to the last good epoch (LGE) — at least to the epoch that is current when you execute MAKE_AHM_NOW.

Syntax

```
MAKE_AHM_NOW ( [ true ] )
```

Parameters

<i>true</i>	<p>Allows AHM to advance when one of the conditions is true:</p> <ul style="list-style-type: none">• One or more nodes are down.• One projection is being refreshed from another (retentive refresh). <p>In either case, you must supply this argument to MAKE_AHM_NOW, otherwise Vertica returns an error. If you execute MAKE_AHM_NOW(<i>true</i>) during retentive refresh, Vertica rolls back the refresh operation and advances the AHM.</p>
-------------	--

Privileges

Superuser

Setting AHM When Nodes Are Down

If you run `MAKE_AHM_NOW` while any node is down, you must supply an argument of `true`; otherwise, Vertica returns an error. In the following example, `MAKE_AHM_NOW` advances the AHM even though a node is down:

```
=> SELECT MAKE_AHM_NOW(true);
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in set AHM
        MAKE_AHM_NOW
-----
        AHM set (New AHM Epoch: 684)
(1 row)
```

Caution: If the AHM is advanced beyond the last good epoch of the failed nodes, those nodes must recover all data from scratch.

See Also

- [SET_AHM_EPOCH](#)
- [SET_AHM_TIME](#)

SET_AHM_EPOCH

Sets the Ancient History Mark (AHM) to the specified epoch. This function allows deleted data up to and including the AHM epoch to be purged from physical storage.

`SET_AHM_EPOCH` is normally used for testing purposes. Instead, consider using [SET_AHM_TIME](#) which is easier to use.

Syntax

```
SET_AHM_EPOCH ( epoch, [ true ] )
```

Parameters

<i>epoch</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> • The number of the epoch in which to set the AHM • Zero (0) (the default) disables PURGE
<i>true</i>	<p>Allows the AHM to advance when nodes are down.</p> <p>Note: If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch. Use with care.</p>

Privileges

Superuser

Restrictions

The number of the specified epoch must be:

- Greater than the current AHM epoch
- Less than the current epoch
- Less than or equal to the cluster last good epoch (the minimum of the last good epochs of the individual nodes in the cluster)

Use the [SYSTEM](#) table to see current values of various epochs related to the AHM, For example:

```
=> SELECT * from SYSTEM;
-[ RECORD 1 ]-----+-----
current_timestamp    | 2009-08-11 17:09:54.651413
current_epoch        | 1512
ahm_epoch            | 961
last_good_epoch      | 1510
refresh_epoch        | -1
designed_fault_tolerance | 1
node_count           | 4
node_down_count      | 0
current_fault_tolerance | 1
catalog_revision_number | 1590
wos_used_bytes       | 0
wos_row_count        | 0
```

```
ros_used_bytes      | 41490783
ros_row_count       | 1298104
total_used_bytes    | 41490783
total_row_count     | 1298104
```

All nodes must be up. You cannot use `SET_AHM_EPOCH` when any node in the cluster is down, except by using the optional `true` parameter.

When a node is down and you use `MAKE_AHM_NOW`, the following error is printed to the `vertica.log`:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they will perform full recovery.
```

Examples

The following command sets the AHM to a specified epoch of 12:

```
=> SELECT SET_AHM_EPOCH(12);
```

The following command sets the AHM to a specified epoch of 2 and allows the AHM to advance despite a failed node:

```
=> SELECT SET_AHM_EPOCH(2, true);
```

See Also

- [MAKE_AHM_NOW](#)
- [SET_AHM_TIME](#)
- [SYSTEM](#)

SET_AHM_TIME

Sets the Ancient History Mark (AHM) to the epoch corresponding to the specified time on the initiator node. This function allows historical data up to and including the AHM epoch to be purged from physical storage.

Syntax

```
SET_AHM_TIME ( time , [ true ] )
```

Parameters

<i>time</i>	Is a TIMESTAMP/TIMESTAMPTZ value that is automatically converted to the appropriate epoch number.
<i>true</i>	[Optional] Allows the AHM to advance when nodes are down. Note: If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch.

Privileges

Superuser

Notes

- SET_AHM_TIME returns a `TIMESTAMP WITH TIME ZONE` value representing the end point of the AHM epoch.
- You cannot change the AHM when any node in the cluster is down, except by using the optional *true* parameter.
- When a node is down and you issue `SELECT MAKE_AHM_NOW()`, the following error is printed to the `vertica.log`:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they will perform full recovery.
```

Examples

Epochs depend on a configured epoch advancement interval. If an epoch includes a three-minute range of time, the purge operation is accurate only to within minus three minutes of the specified timestamp:

```
=> SELECT SET_AHM_TIME('2008-02-27 18:13');
       set_ahm_time
-----
AHM set to '2008-02-27 18:11:50-05'
(1 row)
```

Note: The `-05` part of the output string is a time zone value, an offset in hours from UTC (Universal Coordinated Time, traditionally known as Greenwich Mean Time, or GMT).

In the previous example, the actual AHM epoch ends at 18:11:50, roughly one minute before the specified timestamp. This is because `SET_AHM_TIME` selects the epoch that ends at or before the specified timestamp. It does not select the epoch that ends after the specified timestamp because that would purge data deleted as much as three minutes after the AHM.

For example, using only hours and minutes, suppose that epoch 9000 runs from 08:50 to 11:50 and epoch 9001 runs from 11:50 to 15:50. `SET_AHM_TIME('11:51')` chooses epoch 9000 because it ends roughly one minute before the specified timestamp.

In the next example, suppose that a node went down at 11:00:00 AM on January 1st 2017. At noon, you want to advance the AHM to 11:15:00, but the node is still down.

Suppose you try to set the AHM using this command:

```
=> SELECT SET_AHM_TIME('2017-01-01 11:15:00');
```

Then you will receive an error message. Vertica prevents you from moving the AHM past the point where a node went down. Vertica returns this error to prevent the AHM from advancing past the down node's last good epoch. You can force the AHM to advance by supplying the optional second parameter:

```
=> SELECT SET_AHM_TIME('2017-01-01 11:15:00', true);
```

However, if you force the AHM past the last good epoch, the failed node will have to recover from scratch.

See Also

- [MAKE_AHM_NOW](#)
- [SET_AHM_EPOCH](#)
- [SET DATESTYLE](#)
- [TIMESTAMP/TIMESTAMPTZ](#)

Flex Table Functions

This section contains helper functions for use in working with flex tables.

Note: While the functions are available to all users, they are applicable only to flex table, their associated *flex_table_keys* table and *flex_table_view* views. By computing keys and creating views from flex table data, the functions facilitate SELECT queries. One function restores the original keys table and view that were made when you first created the flex table. For more information, see [Using Flex Tables](#).

BUILD_FLEXTABLE_VIEW

Creates, or re-creates, a view for a default or user-defined *_keys* table, ignoring any empty keys.

Syntax

```
build_flextable_view('flex_table' [ [, 'view_name'] [, 'user_keys_
table'] ])
```

Arguments

<i>flex_table</i>	The flex table name. By default, this function builds or rebuilds a view for the input table with the current contents of the associated <i>flex_table_keys</i> table.
<i>view_name</i>	[Optional] A custom view name. Use this option to build a new view for <i>flex_table</i> with the name you specify.
<i>user_keys_table</i>	[Optional] Specifies a keys table from which to create the view. Use this option if you created a custom <i>user_keys</i> table from the flex table map data, rather than from the default <i>flex_table_keys</i> table. The function builds a view from the keys in <i>user_keys</i> table, rather than from the <i>flex_table_keys</i> table.

Examples

The following examples show how to call `build_flextable_view` with 1, 2, or 3 arguments.

Creating a Default View

To create, or re-create, a default view:

1. Call the function with an input flex table, darkdata:

```
=> SELECT build_flextable_view('darkdata');
           build_flextable_view
-----
The view public.darkdata_view is ready for querying
(1 row)
```

The function creates a view with the default name (darkdata_view) from the darkdata_keys table.

2. Query a key name (user.id) from the new or updated view:

```
=> SELECT "user.id" from darkdata_view;
       user.id
-----
340857907
727774963
390498773
288187825
164464905
125434448
601328899
352494946
(12 rows)
```

Creating a Custom Name View

To create, or re-create, a view with a custom name:

1. Call the function with two arguments, an input flex table, darkdata, and the name of the view to create, dd_view:

```
=> SELECT build_flextable_view('darkdata', 'dd_view');
           build_flextable_view
-----
The view public.dd_view is ready for querying
(1 row)
```

2. Query a key name (user.lang) from the new or updated view (dd_view):

```
=> SELECT "user.lang" from dd_view;
       user.lang
-----
tr
en
es
en
en
it
```

```
es
en
(12 rows)
```

Creating a View from a Custom Keys Table

To create a view from a custom `_keys` table with `build_flextable_view`, the custom table must have the same schema and table definition as the default table (`darkdata_keys`).

Create a custom keys table, using any of these three approaches:

1. Create a columnar table with all keys from the default keys table for a flex table (`darkdata_keys`):

```
=> CREATE table new_darkdata_keys as select * from darkdata_keys;
CREATE TABLE
```

2. Create a columnar table without content (`LIMIT 0`) from the default keys table for a flex table (`darkdata_keys`):

```
=> CREATE table new_darkdata_keys as select * from darkdata_keys LIMIT 0;
CREATE TABLE
kdb=> select * from new_darkdata_keys;
 key_name | frequency | data_type_guess
-----+-----+-----
(0 rows)
```

3. Create a columnar table without content (`LIMIT 0`) from the default keys table, and insert two values (`'user.lang'`, `'user.name'`) into the `key_name` column:

```
=> CREATE table dd_keys as select * from darkdata_keys limit 0;
CREATE TABLE
=> insert into dd_keys (key_name) values ('user.lang');
OUTPUT
-----
      1
(1 row)
=> INSERT into dd_keys (key_name) values ('user.name');
OUTPUT
-----
      1
(1 row)
=> SELECT * from dd_keys;
 key_name | frequency | data_type_guess
-----+-----+-----
 user.lang |          | 
 user.name |          | 
(2 rows)
```

4. After creating a custom keys table, call `build_flextable_view` with all arguments (an input flex table, the new view name, the custom keys table):

```
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata', 'dd_view', 'dd_keys');
       build_flextable_view
-----
The view public.dd_view is ready for querying
(1 row)
```

5. Query the new view:

```
=> SELECT * from dd_view;
```

See Also

- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [MATERIALIZED_FLEXTABLE_COLUMNS](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

COMPUTE_FLEXTABLE_KEYS

Computes the virtual columns (keys and values) from the flex table VMap data. Use this function to compute keys without creating an associated table view. To also build a view, use [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#).

The function stores its results in the associated flex `_keys` table, which has the following columns:

- `key_name`
- `frequency`
- `data_type_guess`

For more information, see [Computing Flex Table Keys](#).

Syntax

```
compute_flextable_keys('flex_table')
```

Arguments

<i>flex_table</i>	The name of a flex table.
-------------------	---------------------------

Using Data Type Guessing

The results of the `flex_keys` table `data_type_guess` column depend on the `EnableBetterFlexTypeGuessing` configuration parameter. By default, the parameter is 1 (ON). This setting results in the function returning all non-string keys in the `data_type_guess` column as one of the following types (and others listed in [SQL Data Types](#)):

- BOOLEAN
- INTEGER
- FLOAT
- TIMESTAMP
- DATE

Setting the configuration parameter to 0 (OFF), results in the function returning only string types (`[LONG]VARCHAR`) or (`[LONG] VARBINARY`) for all values in the `data_type_guess` column of the `flex_keys` table .

Assigning Flex Key Data Types

Use the sample CSV data in this section to compare the results of using or not using the `EnableBetterFlexTypeGuessing` configuration parameter. When the parameter is ON, the function determines key non-string data types in your map data more accurately. The default for the parameter is 1 (ON).

```
Year,Quarter,Region,Species,Grade,Pond Value,Number of Quotes,Available
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,1P,$615.12 ,12,No
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,SM,$610.78 ,12,Yes
2015,1,2 - Northwest Oregon & Willamette,Douglas-fir,2S,$596.00 ,20,Yes
2015,1,2 - Northwest Oregon & Willamette,Hemlock,P,$520.00 ,6,Yes
2015,1,2 - Northwest Oregon & Willamette,Hemlock,SM,$510.00 ,6,No
2015,1,2 - Northwest Oregon & Willamette,Hemlock,2S,$490.00 ,14,No
```

To compare the data type assignment results, complete the following steps:

1. Save the CSV data file (here, as `trees.csv`).
2. Create a flex table (`trees`) and load `trees.csv` using the `fcsvparser`:

```
=> CREATE FLEX TABLE trees();  
=> COPY trees FROM '/home/dbadmin/tempdat/trees.csv' PARSER fcsvparser();
```

3. Use `COMPUTE_FLEXTABLE_KEYS` with the `trees` flex table.

```
=> SELECT COMPUTE_FLEXTABLE_KEYS('trees');  
          COMPUTE_FLEXTABLE_KEYS  
-----  
Please see public.trees_keys for updated keys  
(1 row)
```

4. Query the `trees_keys` table output.:

```
=> SELECT * FROM trees_keys;  
   key_name | frequency | data_type_guess  
-----+-----+-----  
Year        |          6 | Integer  
Quarter     |          6 | Integer  
Region      |          6 | Varchar(66)  
Available   |          6 | Boolean  
Number of Quotes |          6 | Integer  
Grade       |          6 | Varchar(20)  
Species     |          6 | Varchar(22)  
Pond Value  |          6 | Numeric(8,3)  
(8 rows)
```

5. Set the `EnableBetterFlexTypeGuessing` parameter to 0 (OFF).
6. Call `COMPUTE_FLEXTABLE_KEYS` with the `trees` flex table again.
7. Query the `trees_keys` table to compare the `data_type_guess` values with the previous results. Without the configuration parameter set, all of the non-string data types are `VARCHARS` of various lengths:

```
=> SELECT * FROM trees_keys;  
   key_name | frequency | data_type_guess  
-----+-----+-----  
Year        |          6 | varchar(20)  
Quarter     |          6 | varchar(20)  
Region      |          6 | varchar(66)  
Available   |          6 | varchar(20)  
Grade       |          6 | varchar(20)  
Number of Quotes |          6 | varchar(20)  
Pond Value  |          6 | varchar(20)  
Species     |          6 | varchar(22)
```

(8 rows)

8. To maintain accurate results for non-string data types, set the `EnableBetterFlexTypeGuessing` parameter back to 1 (ON).

For more information about setting the `EnableBetterFlexTypeGuessing` configuration parameter, see [Setting Flex Table Configuration Parameters](#).

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [MATERIALIZATE_FLEXTABLE_COLUMNS](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW

Combines the functionality of [BUILD_FLEXTABLE_VIEW](#) and [COMPUTE_FLEXTABLE_KEYS](#) to compute virtual columns (keys) from the VMap data of a flex table and construct a view. Creating a view with this function ignores empty keys. If you do not need to perform both operations together, use one of the single-operation functions instead.

Syntax

```
compute_flextable_keys_and_build_view('flex_table')
```

Arguments

<i>flex_table</i>	The name of a flex table.
-------------------	---------------------------

Examples

This example shows how to call the function for the `darkdata` flex table.

```
=> SELECT compute_flextable_keys_and_build_view('darkdata');  
        compute_flextable_keys_and_build_view  
-----
```

```
Please see public.darkdata_keys for updated keys
The view public.darkdata_view is ready for querying
(1 row)
```

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS](#)
- [MATERIALIZE_FLEXTABLE_COLUMNS](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

EMPTYMAP

Constructs a new VMap with one row but without keys or data. Use this transform function to populate a map without using a flex parser. Instead, you use either from SQL queries or from map data present elsewhere in the database.

Syntax

```
emptymap()
```

Arguments

None

Examples

Create an Empty Map

```
=> SELECT emptymap();
           emptymap
-----
\001\000\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

Create an Empty Map from an Existing Flex Table

If you create an empty map from an existing flex table, the new map has the same number of rows as the table from which it was created.

clause for the source table.

Syntax

```
mapaggregate(source_column1, source_column2)
```

Arguments

<code>source_column1</code>	Table column with values to use as the keys of the key/value pair of the returned VMap data.
<code>source_column2</code>	Table column with values to use as the values in the key/value pair of the returned VMap data.

Examples

This example creates a columnar table `btest`, with two VARCHAR columns, named `keys` and `values`, and adds three sets of values:

```
=> CREATE table btest(keys varchar(10), values varchar(10));
CREATE TABLE
=> copy btest from stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> one|1
>> two|2
>> three|3
>> \.
```

After populating the `btest` table, call `mapaggregate()`, using the `over` (`PARTITION BEST`) clause. This call returns the `raw_map` data:

```
=> SELECT mapaggregate(keys, values) over(PARTITION BEST) from btest;
          raw_map
-----
\001\000\000\000\023\000\000\000\003\000\000\000\020\000\000\000\021\000\000\000\022\000\000\000\0132
\003\000\000\000\020\000\000\000\023\000\000\000\030\000\000\000onethreetwo
(1 row)
```

The next example illustrates using `MAPTOSTRING()` with the returned `raw_map` from `mapaggregate()` to see the values:

```
=> SELECT maptostring(raw_map) from (select mapaggregate(keys, values) over(PARTITION BEST) from
btest) bit;
          maptostring
-----
```

```
{
  "one": "1",
  "three": "3",
  "two": "2"
}
(1 row)
```

See Also

- [EMPTYMAP](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPCONTAINSKEY

Determines whether a VMap contains a virtual column (key). This scalar function returns true (t), if the virtual column exists, or false (f) if it does not. Determining that a key exists before calling `maplookup()` lets you distinguish between NULL returns. The `maplookup()` function uses for both a non-existent key and an existing key with a NULL value.

Syntax

```
mapcontainskey(VMap_data, 'virtual_column_name')
```

Arguments

VMap_data	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none"> • The <code>__raw__</code> column of a flex table • Data returned from a map function such as <code>maplookup()</code> • Other database content
virtual_column_name	The name of the key to check.

Examples

This example shows how to use the `mapcontainskey()` functions with `maplookup()`. View the results returned from both functions. Check whether the empty fields that `maplookup()` returns indicate a NULL value for the row (t) or no value (f):

You can use `mapcontainskey()` to determine that a key exists before calling `maplookup()`. The `maplookup()` function uses both NULL returns and existing keys with NULL values to indicate a non-existent key.

```
=> SELECT maplookup(__raw__, 'user.location'), mapcontainskey(__raw__, 'user.location') from darkdata
order by 1;
 maplookup | mapcontainskey
-----+-----
          | t
          | t
          | t
          | t
Chile     | t
Narnia    | t
Uptown.. | t
chicago  | t
          | f
          | f
          | f
          | f
(12 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPCONTAINSVALUE

Determines whether a VMap contains a specific value. Use this scalar function to return true (t), if the value exists, or false (f), if it does not.

Syntax

```
mapcontainsvalue(VMap_data, 'virtual_column_value')
```

Arguments

VMap_data	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
-----------	---

virtual_column_value	The value whose existence you want to confirm.
----------------------	--

Examples

This example shows how to use `mapcontainsvalue()` to determine whether or not a virtual column contains a particular value. Create a flex table (`ftest`), and populate it with some virtual columns and values. Name both virtual columns one:

```
=> CREATE flex table ftest();
CREATE TABLE
=> copy ftest from stdin parser fjsonparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> {"one":1, "two":2}
>> {"one":"one", "2":"2"}
>> \.
```

Call `mapcontainsvalue()` on the `ftest` map data. The query returns false (`f`) for the first virtual column, and true (`t`) for the second, which contains the value one:

```
=> SELECT mapcontainsvalue(__raw__, 'one') from ftest;
mapcontainsvalue
-----
f
t
(2 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)

- [MAPVALUES](#)
- [MAPVERSION](#)

MAPDELIMITEDEXTRACTOR

Extracts data with a delimiter character, and other optional arguments, returning a single VMap value. The USING PARAMETERS phrase specifies optional parameters for the function.

Parameters

delimiter	VARCHAR	Single delimiter character. Default value:
header_names	VARCHAR	[Optional] Specifies header names for columns. Default value: uco1n Where <i>n</i> is the column offset number, starting with 0 for the first column. The function uses default values if you do not specify values for the header_names parameter.
trim	BOOLEAN	[Optional] Trims white space from header names and field values. Default value: true
treat_empty_val_as_null	BOOLEAN	[Optional] Specifies that empty fields become NULLs, rather than empty strings (' '). Default value: true

Examples

These examples use a short set of delimited data:

```
Name|CITY|New city|State|zip
Tom|BOSTON|boston|MA|01
Eric|Burlington|BURLINGTON|MA|02
Jamie|cambridge|CAMBRIDGE|MA|08
```

To begin, save this data as `delim.dat`.

1. Create a flex table, `dflex`:

```
=> CREATE flex table dflex();  
CREATE TABLE
```

2. Use `COPY` to load the `delim.dat` file. Use the flex tables `fdelimitedparser` with the `header='false'` option:

```
=> COPY dflex from '/home/release/kmm/flextables/delim.dat' parser fdelimitedparser  
(header='false');  
Rows Loaded  
-----  
4  
(1 row)
```

3. Create a columnar table, `dtab`, with an identity `id` column, a `delim` column, and a column to hold a `VMap`, named `vmap`:

```
=> CREATE table dtab (id IDENTITY(1,1), delim varchar(128), vmap long varbinary(512));  
CREATE TABLE
```

4. Use `COPY` to load the `delim.dat` file into the `dtab` table. For the `mapdelimitedextractor` function, add a header row with `USING PARAMETERS header_names=` option to specify the header row for the sample data, along with `delimiter '!'`:

```
=> COPY dtab(delim, vmap as mapdelimitedextractor(delim  
USING PARAMETERS header_names='Name|CITY|New City|State|Zip')) FROM  
'/home/dbadmin/data/delim.dat' DELIMITER '!';  
Rows Loaded  
-----  
4  
(1 row)
```

5. Use `maptostring` for the flex table `dflex` to view the `__raw__` column contents. Notice the default header names in use (`ucol0 – ucol4`), since you specified `header='false'` when you loaded the flex table:

```
=> SELECT maptostring(__raw__) from dflex limit 10;  
maptostring  
-----  
{  
  "ucol0" : "Jamie",  
  "ucol1" : "cambridge",  
  "ucol2" : "CAMBRIDGE",  
  "ucol3" : "MA",  
  "ucol4" : "08"
```

```
}  
  
{  
  "ucol0" : "Name",  
  "ucol1" : "CITY",  
  "ucol2" : "New city",  
  "ucol3" : "State",  
  "ucol4" : "zip"  
}  
  
{  
  "ucol0" : "Tom",  
  "ucol1" : "BOSTON",  
  "ucol2" : "boston",  
  "ucol3" : "MA",  
  "ucol4" : "01"  
}  
  
{  
  "ucol0" : "Eric",  
  "ucol1" : "Burlington",  
  "ucol2" : "BURLINGTON",  
  "ucol3" : "MA",  
  "ucol4" : "02"  
}  
  
(4 rows)
```

6. Use `maptostring` again, this time with the `dtab` table's `vmap` column. Compare the results of this output to those for the `flex` table. Note that `maptostring` returns the `header_name` parameter values you specified when you loaded the data:

```
=> SELECT maptostring(vmap) from dtab;  
  
                                maptostring  
-----  
  
{  
  "CITY" : "CITY",  
  "Name" : "Name",  
  "New City" : "New city",  
  "State" : "State",  
  "Zip" : "zip"  
}  
  
{  
  "CITY" : "BOSTON",  
  "Name" : "Tom",  
  "New City" : "boston",  
  "State" : "MA",  
  "Zip" : "02121"  
}  
  
{  
  "CITY" : "Burlington",  
  "Name" : "Eric",
```

```
"New City" : "BURLINGTON",  
"State" : "MA",  
"Zip" : "02482"  
}  
  
{  
  "CITY" : "cambridge",  
  "Name" : "Jamie",  
  "New City" : "CAMBRIDGE",  
  "State" : "MA",  
  "Zip" : "02811"  
}  
  
(4 rows)
```

7. Query the `delim` column to view the contents differently:

```
=> SELECT delim from dtab;  
      delim  
-----  
Name|CITY|New city|State|zip  
Tom|BOSTON|boston|MA|02121  
Eric|Burlington|BURLINGTON|MA|02482  
Jamie|cambridge|CAMBRIDGE|MA|02811  
(4 rows)
```

See Also

- [MAPJSONEXTRACTOR](#)
- [MAPREGEXEXTRACTOR](#)

MAPITEMS

Returns information about items in a VMap. Use this transform function with one or more optional arguments to access polystructured values within the VMap data. This function requires an `OVER()` clause.

Syntax

```
mapItems(VMap_data [, passthrough_arg [,... ] ])
```

Arguments

<code>VMap_data</code>	Any VMap data. The VMap can exist as:
------------------------	---------------------------------------

	<ul style="list-style-type: none"> • The <code>__raw__</code> column of a flex table • Data returned from a map function such as <code>maplookup()</code> • Other database content
<code>passthrough_arg</code>	[Optional] One or more arguments indicating keys within the map data in <code>VMap_data</code>

Examples

The following examples illustrate using `mapItems()` with the `over(PARTITION BEST)` clause.

This example determines the number of virtual columns in the map data using a flex table, labeled `darkmountain`. Query using the `count()` function to return the number of virtual columns in the map data:

```
=> SELECT count(keys) FROM (select mapitems(darkmountain.__raw__) over(PARTITION BEST) from
darkmountain) as a;
count
-----
      19
(1 row)
```

The next example determines what items exist in the map data:

```
=> SELECT * FROM (select mapitems(darkmountain.__raw__) over(PARTITION BEST) from darkmountain) as a;
keys      | values
-----+-----
hike_safety | 50.6
name       | Mt Washington
type       | mountain
height    | 17000
hike_safety | 12.2
name       | Denali
type       | mountain
height    | 29029
hike_safety | 34.1
name       | Everest
type       | mountain
height    | 14000
hike_safety | 22.8
name       | Kilimanjaro
type       | mountain
height    | 29029
hike_safety | 15.4
name       | Mt St Helens
type       | volcano
(19 rows)
```

Directly Query a Key Value in a VMap

Review the following JSON input file, `simple.json`. In particular, notice the array called `three_Array`, and its four values:

```
{
  "one": "one",
  "two": 2,
  "three_Array":
  [
    "three_One",
    "three_Two",
    3,
    "three_Four"
  ],
  "four": 4,
  "five_Map":
  {
    "five_One": 51,
    "five_Two": "Fifty-two",
    "five_Three": "fifty three",
    "five_Four": 54,
    "five_Five": "5 x 5"
  },
  "six": 6
}
```

1. Create a flex table, mapper:

```
=> CREATE flex table mapper();
CREATE TABLE
```

1. Load `simple.json` into the flex table mapper:

```
=> COPY mapper from '/home/dbadmin/data/simple.json' parser fjsonparser (flatten_arrays=false,
flatten_maps=false);
Rows Loaded
-----
                1
(1 row)
```

2. Call `mapkeys` on the flex table's `__raw__` column to see the flex table's keys, but not the key submaps. The return values indicate `three_Array` as one of the virtual columns:

```
=> SELECT mapkeys(__raw__) over() from mapper;
keys
-----
five_Map
four
one
six
three_Array
two
(6 rows)
```

3. Call `mapitems` on flex table mapper with `three_Array` as a pass-through argument to the function. The call returns these array values:

```
=> SELECT __identity__, mapitems(three_Array) OVER(PARTITION BY __identity__) from mapper;
__identity__ | keys | values
-----+-----+-----
          1 | 0    | three_One
          1 | 1    | three_Two
          1 | 2    | 3
          1 | 3    | three_Four
(4 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPJSONEXTRACTOR

Extracts content of repeated JSON data objects, including nested maps, or data with an outer list of JSON elements. The `USING PARAMETERS` phrase specifies optional parameters for the function. Empty input does not generate a Warning or Error.

Note: The function fails if the output size of the function is greater than 65000.

Parameters

<code>flatten_maps</code>	BOOLEAN	[Optional] Flattens sub-maps within the JSON data, separating map levels with a period (.). Default value: true
<code>flatten_arrays</code>	BOOLEAN	[Optional] Converts lists to sub-maps with integer keys. Lists are not flattened by default. Default value: false
<code>reject_on_duplicate</code>	BOOLEAN	[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues. Default value: false
<code>reject_on_empty_key</code>	BOOLEAN	[Optional] Rejects any row containing a key without a value (<code>reject_on_empty_key=true</code>). Default value: false
<code>omit_empty_keys</code>	BOOLEAN	[Optional] Omits any key from the load data that does not have a value (<code>omit_empty_keys=true</code>). Default value: false
<code>start_point</code>	CHAR	[Optional] Specifies the name of a key in the JSON load data at which to begin parsing. The parser ignores all data before the <code>start_point</code> value. The parser processes data after the first instance, and up to the second, ignoring any remaining data. Default value: none

Examples

These examples use the following sample JSON data:

```
{ "id": "5001", "type": "None" }
{ "id": "5002", "type": "Glazed" }
{ "id": "5005", "type": "Sugar" }
{ "id": "5007", "type": "Powdered Sugar" }
{ "id": "5004", "type": "Maple" }
```

Save this example data as `bake_single.json`, and load that file.

1. Create a flex table, flexjson:

```
=> CREATE flex table flexjson();  
CREATE TABLE
```

2. Use COPY to load the bake_single.json file with the fjsonparser parser:

```
=> COPY flexjson from '/home/dbadmin/data/bake_single.json' parser fjsonparser();  
Rows Loaded  
-----  
5  
(1 row)
```

3. Create a columnar table, coljson, with an identity column (id), a json column, and a column to hold a VMap, called vmap:

```
=> CREATE table coljson(id IDENTITY(1,1), json varchar(128), vmap long varbinary(10000));  
CREATE TABLE
```

4. Use COPY to load the bake_single.json file into the coljson table, using the mapjsonextractor function:

```
=> COPY coljson (json, vmap AS MapJSONExtractor(json)) FROM '/home/dbadmin/data/bake_  
single.json';  
Rows Loaded  
-----  
5  
(1 row)
```

5. Use the maptostring function for the flex table flexjson to output the __raw__ column contents as strings:

```
=> SELECT maptostring(__raw__) from flexjson limit 5;  
maptostring  
-----  
{  
  "id" : "5001",  
  "type" : "None"  
}  
  
{  
  "id" : "5002",  
  "type" : "Glazed"  
}  
  
{  
  "id" : "5005",  
  "type" : "Sugar"  
}
```

```
{
  "id" : "5007",
  "type" : "Powdered Sugar"
}

{
  "id" : "5004",
  "type" : "Maple"
}

(5 rows)
```

6. Use the `maptostring` function again, this time with the `coljson` table's `vmap` column and compare the results. The element order differs:

```
=> SELECT maptostring(vmap) from coljson limit 5;
      maptostring
-----
{
  "id" : "5001",
  "type" : "None"
}

{
  "id" : "5002",
  "type" : "Glazed"
}

{
  "id" : "5004",
  "type" : "Maple"
}

{
  "id" : "5005",
  "type" : "Sugar"
}

{
  "id" : "5007",
  "type" : "Powdered Sugar"
}

(5 rows)
```

See Also

- [MAPDELIMITEDEXTRACTOR](#)
- [MAPREGEXEXTRACTOR](#)

MAPKEYS

Returns the virtual columns (and values) present in any VMap data. This transform function requires an `over(PARTITION BEST)` clause.

Syntax

```
mapkeys(VMap_data)
```

Arguments

<code>VMap_data</code>	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
------------------------	--

Examples

Determine Number of Virtual Columns in Map Data

This example shows how to create a query, using an `over(PARTITION BEST)` clause with a flex table, `darkdata` to find the number of virtual column in the map data. The table is populated with JSON tweet data.

```
=> SELECT count(keys) FROM (SELECT mapkeys(darkdata.__raw__) over(PARTITION BEST) from darkdata) as a;  
count  
-----  
550  
(1 row)
```

Query Ordered List of All Virtual Columns in the Map

This example shows a snippet of the return data when you query an ordered list of all virtual columns in the map data:

```
=> SELECT * FROM (SELECT mapkeys(darkdata.__raw__) over(PARTITION BEST) from darkdata) as a;  
keys  
-----
```

```
contributors
coordinates
created_at
delete.status.id
delete.status.id_str
delete.status.user_id
delete.status.user_id_str
entities.hashtags
entities.media
entities.urls
entities.user_mentions
favorited
geo
id
.
.
.
user.statuses_count
user.time_zone
user.url
user.utc_offset
user.verified
(125 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPKEYSINFO

Returns virtual column information from a given map. This transform function requires an `over(PARTITION BEST)` clause.

Syntax

```
mapkeysinfo(VMap_data)
```

Arguments

<code>VMap_data</code>	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
------------------------	--

Returns

This function is a superset of the [MAPKEYS\(\)](#) function. It returns the following information about each virtual column:

Column	Description
<code>keys</code>	The virtual column names in the raw data.
<code>length</code>	The data length of the key name, which can differ from the actual string length.
<code>type_oid</code>	The OID type into which the value should be converted. Currently, the type is always 116 for a <code>LONG VARCHAR</code> , or 199 for a nested map that is stored as a <code>LONG VARBINARY</code> .
<code>row_num</code>	The number of rows in which the key was found.
<code>field_num</code>	The field number in which the key exists.

Examples

This example shows a snippet of the return data you receive if you query an ordered list of all virtual columns in the map data:

```
=> SELECT * FROM (SELECT mapkeysinfo(darkdata.__raw__) over(PARTITION BEST) from darkdata) as a;
      keys | length | type_oid | row_num | field_num
-----+-----+-----+-----+-----
contributors | 0 | 116 | 1 | 0
coordinates | 0 | 116 | 1 | 1
created_at | 30 | 116 | 1 | 2
entities.hashtags | 93 | 199 | 1 | 3
entities.media | 772 | 199 | 1 | 4
entities.urls | 16 | 199 | 1 | 5
entities.user_mentions | 16 | 199 | 1 | 6
favorited | 1 | 116 | 1 | 7
geo | 0 | 116 | 1 | 8
id | 18 | 116 | 1 | 9
id_str | 18 | 116 | 1 | 10
.
.
.
delete.status.id | 18 | 116 | 11 | 0
delete.status.id_str | 18 | 116 | 11 | 1
delete.status.user_id | 9 | 116 | 11 | 2
delete.status.user_id_str | 9 | 116 | 11 | 3
delete.status.id | 18 | 116 | 12 | 0
delete.status.id_str | 18 | 116 | 12 | 1
delete.status.user_id | 9 | 116 | 12 | 2
delete.status.user_id_str | 9 | 116 | 12 | 3
(550 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)

- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPLOOKUP

Returns single-key values from VMAP data. This scalar function returns a LONG VARCHAR, with values, or NULL if the virtual column does not have a value.

Using `maplookup` is case insensitive to virtual column names. To avoid loading same-name values, set the `fjsonparser parser reject_on_duplicate` parameter to `true` when data loading.

You can control the behavior for non-scalar values in a VMAP (like arrays), when loading data with the `fjsonparser` or `favroparser` parsers and its `flatten-arrays` argument. See [Loading JSON Data](#) and the [FJSONPARSER](#) reference.

For information about using `maplookup()` to access nested JSON data, see [Querying Nested Data](#).

Syntax

```
maplookup (VMap_data, 'virtual_column_name' [USING PARAMETERS [case_sensitive={false | true}] [, buffer_size=n] ] )
```

Parameters

<code>VMap_data</code>	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
<code>virtual_column_name</code>	The name of the virtual column whose values this function returns.
<code>buffer_size</code>	[Optional parameter] Specifies the maximum length (in bytes) of each value returned for <code>virtual_column_name</code> . To return all values for <code>virtual_column_name</code> , specify a <code>buffer_size</code> equal to or greater than (<code>=></code>) the number of bytes for any returned value. Any returned

	values greater in length than <code>buffer_size</code> are rejected. Default value: 0 (No limit on <code>buffer_size</code>)
<code>case_sensitive</code>	[Optional parameter] Specifies whether to return values for <code>virtual_column_name</code> if keys with different cases exist. Example: (... USING PARAMETERS <code>case_sensitive=true</code>) Default value: false

Examples

This example returns the values of one virtual column, `user.location`:

```
=> SELECT maplookup(__raw__, 'user.location') FROM darkdata order by 1;
maplookup
-----
Chile
Nesnia
Uptown
.
.
chicago
(12 rows)
```

Using `maplookup buffer_size`

Use the `buffer_size=` parameter to indicate the maximum length of any value that `maplookup` returns for the virtual column you specify. If none of the returned key values can be greater than `n` bytes, use this parameter to allocate `n` bytes as the `buffer_size`.

For the next example, save this JSON data to a file, `simple_name.json`:

```
{
  "name": "sierra",
  "age": "63",
  "eyes": "brown",
  "weapon": "doggie"
}
{
  "name": "janis",
  "age": "10",
  "eyes": "blue",
  "weapon": "humor"
}
{
  "name": "ben",
  "age": "43",
```

```
"eyes": "blue",  
"weapon": "sword"  
}  
{  
  "name": "jen",  
  "age": "38",  
  "eyes": "green",  
  "weapon": "shopping"  
}
```

1. Create a flex table, logs.
2. Load the simple_name.json data into logs, using the fjsonparser. Specify the flatten_arrays option as True:

```
=> COPY logs FROM '/home/dbadmin/data/simple_name.json'  
  PARSER fjsonparser(flatten_arrays=True);
```

3. Use maplookup with buffer_size=0 for the logs table name key. This query returns all of the values:

```
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=0) FROM logs;  
MapLookup  
-----  
sierra  
ben  
janis  
jen  
(4 rows)
```

4. Next, call maplookup() three times, specifying the buffer_size parameter as 3, 5, and 6, respectively. Now, maplookup() returns values with a byte length less than or equal to (<=) buffer_size:

```
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=3) FROM logs;  
MapLookup  
-----  
  
ben  
  
jen  
(4 rows)  
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=5) FROM logs;  
MapLookup  
-----  
  
janis  
jen  
ben  
(4 rows)
```

```
=> SELECT MapLookup(__raw__, 'name' USING PARAMETERS buffer_size=6) FROM logs;
MapLookup
-----
sierra
janis
jen
ben
(4 rows)
```

Disambiguate Empty Output Rows

This example shows how to interpret empty rows. Using `maplookup` without first checking whether a key exists can be ambiguous. When you review the following output, 12 empty rows, you cannot determine whether a `user.location` key has:

- A non-NULL value
- A NULL value
- No value

```
=> SELECT maplookup(__raw__, 'user.location') FROM darkdata;
maplookup
-----
(12 rows)
```

To disambiguate empty output rows, use the `mapcontainskey()` function in conjunction with `maplookup()`. When `maplookup` returns an empty field, the corresponding value from `mapcontainskey` indicates `t` for a NULL or other value, or `f` for no value.

The following example output using both functions lists rows with NULL or a name value as `t`, and rows with no value as `f`:

```
=> SELECT maplookup(__raw__, 'user.location'), mapcontainskey(__raw__, 'user.location')
from darkdata order by 1;
maplookup | mapcontainskey
-----+-----
          | t
          | t
          | t
```



```
=> CREATE flex table dupe();  
CREATE TABLE  
dbt=> copy dupe from '/home/release/KData/repeated_key_name.json' parser fjsonparser();  
Rows Loaded  
-----  
                8  
(1 row)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPPUT

Accepts a VMap and one or more key/value pairs and returns a new VMap with the key/value pairs added. Keys must be set using the auxiliary function `SetMapKeys()`, and can only be constant strings. If the VMap has any of the new input keys, then the original values are replaced by the new ones.

Syntax

`MapPut(VMap_data, value1 [, value2, value3, ...]` using parameters
`keys=SetMapKeys('key1'[, 'key2', 'key3', ...])`

Arguments

<code>VMap_data</code>	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
<code>value</code>	One or more values to add to the VMap specified in <code>VMap_data</code> .

Parameters

<code>keys</code>	The <code>keys</code> parameter must be the result of <code>SetMapKeys()</code> . <code>SetMapKeys()</code> simply takes one or more constant strings as arguments.
-------------------	---

The following example shows how to create a flex table and use `COPY` to enter some basic JSON data. After creating a second flex table (`vmapdata2`), insert the new vmap results from `mapput()`, with additional key/value pairs.

1. Create sample table:

```
=> CREATE flex table vmapdata1();  
CREATE TABLE
```

2. Load sample JSON data from STDIN:

```
=> COPY vmapdata1 FROM stdin parser fjsonparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"aaa": 1, "bbb": 2, "ccc": 3}  
>> \.
```

3. Create another flex table and use the function to insert data into it:

```
=> CREATE flex table vmapdata2();  
=> INSERT into vmapdata2 select mapput(__row__, '7','8','9'  
    using parameters keys=SetMapKeys('xxx','yyy','zzz')) from vmapdata1;
```

4. View the difference between the original and the new flex tables:

```
=> SELECT maptostring(__row__) FROM vmapdata1;  
      maptostring  
-----  
{  
  "aaa" : "1",  
  "bbb" : "2",  
  "ccc" : "3"  
}  
(1 row)  
=> select maptostring(__row__) from vmapdata2;  
      maptostring  
-----  
{  
  "mapput" : {  
    "aaa" : "1",  
    "bbb" : "2",  
    "ccc" : "3",  
    "xxx" : "7",  
    "yyy" : "8",  
    "zzz" : "9"  
  }  
}
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)

- [MAPSIZE](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPREGEXEXTRACTOR

Extracts data from a regular expression and returns the results as a VMap. Use the USING PARAMETERS pattern= phrase, followed by the regular expression.

Parameters

pattern=	VARCHAR	The regular expression as a string. Default value: An empty string ("").
use_jit	BOOLEAN	[Optional] Uses just-in-time compiling when parsing the regular expression. Default value: false.
record_terminator	VARCHAR	[Optional] The character used to separate input records. Default value: \n.
logline_column	VARCHAR	[Optional] The destination column containing the full string that the regular expression matched. Default value: An empty string ("").

Examples

These examples use the following regular expression, which searches for information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.

This example expression loads any thread_id hex value, regardless of whether it has a 0x prefix, (<thread_id>(?:0x)?[0-9a-f]+).

```
'^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)
 (?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
 -?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
 \<(?!<level>\w+)\> )?(?:<(?!<elevel>\w+)\> @[(?!<enode>\w+)\>]? : )
 (?<text>.*).'
```

The following examples may include newline characters for display purposes.

1. Create a flex table, flogs:

```
=> CREATE flex table flogs();
CREATE TABLE
```

2. Use COPY to load a sample log file (vertica.log), using the flex table fregexparser. Note that this example includes added line characters for displaying long text lines.

```
=> COPY flogs FROM '/home/dbadmin/tempdat/vertica.log' PARSER FREGEXPARSER(pattern='
^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+) (?<thread_name>[A-Za-z ]+):
(?<thread_id>(?:0x)?[0-9a-f])-(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
\<(?!<level>\w+)\> )?(?:<(?!<elevel>\w+)\> @[(?!<enode>\w+)\>]? : )?(?<text>.*)');
Rows Loaded
-----
81399
(1 row)
```

3. Use MapToString to return the results from calling MapRegexExtractor with a regular expression. The output returns the results of the function in string format.

```
=> SELECT maptostring(MapregexExtractor(E'2014-04-02 04:02:51.011
TM Moveout:0x2aab9000f860-a000000002067 [Txn] <INFO>
Begin Txn: a000000002067 'Moveout: Tuple Mover\'' using PARAMETERS
pattern='^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)
 (?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
 -?(?<transaction_id>[0-9a-f])?(?:[(?<component>\w+)]
 \<(?!<level>\w+)\> )?(?:<(?!<elevel>\w+)\> @[(?!<enode>\w+)\>]? : )
 (?<text>.*)'')) FROM flogs where __identity__=13;

maptostring
-----
--
{
"component" : "Txn",
"level" : "INFO",
"text" : "Begin Txn: a000000002067 'Moveout: Tuple Mover'",
"thread_id" : "0x2aab9000f860",
"thread_name" : "TM Moveout",
"time" : "2014-04-02 04:02:51.011",
"transaction_id" : "a000000002067"
}
(1 row)
```

See Also

- [MAPDELIMITEDEXTRACTOR](#)
- [MAPJSONEXTRACTOR](#)

MAPSIZE

Returns the number of virtual columns present in any VMap data. Use this scalar function to determine the size of keys.

Syntax

```
mapsize(VMap_data)
```

Arguments

VMap_data	<p>Any VMap data. The VMap can exist as:</p> <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
-----------	--

Examples

This example shows the returned sizes from the number of keys in the flex table `darkmountain`:

```
=> SELECT mapsize(__raw__) FROM darkmountain;
mapsize
-----
      3
      4
      4
      4
      4
(5 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)
- [MAPVERSION](#)

MAPTOSTRING

Recursively builds a string representation VMap data, including nested JSON maps. Use this transform function to display the VMap contents in a readable LONG VARCHAR format. Use `maptostring` to see how map data is nested before querying virtual columns with `mapvalues()`.

Syntax

```
maptostring(VMap_data [using parameters canonical_json={true  
| false}])
```

Arguments

VMap_data	Any VMap data. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table
-----------	---

	<ul style="list-style-type: none">• Data returned from a map function such as <code>maplookup()</code>• Other database content
--	---

Parameters

<code>canonical_json</code>	<p><i>=bool</i> [Optional parameter]</p> <p>Produces canonical JSON output by default, using the first instance of any duplicate keys in the map data.</p> <p>Use this parameter as other UDF parameters, preceded by using parameters, as shown in the examples. Setting this argument to <code>false</code> maintains the previous behavior of <code>maptostring()</code> and returns same-name keys and their values.</p> <p>Default value: <code>canonical-json=true</code></p>
-----------------------------	--

Examples

The following example shows how to create a sample flex table, `darkdata` and load JSON data from STDIN. By calling `maptostring()` twice with both values for the `canonical_json` parameter, you can see the different results on the flex table `__raw__` column data.

1. Create sample table:

```
=> CREATE flex table darkdata();  
CREATE TABLE
```

2. Load sample JSON data from STDIN:

```
=> COPY darkdata FROM stdin parser fjsonparser();  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> {"aaa": 1, "aaa": 2, "AAA": 3, "bbb": "aaa\"bbb"}  
>> \.
```

3. Call `maptostring()` with its default behavior using canonical JSON output, and then review the flex table contents. The function returns the first duplicate key and its value (`"aaa": "1"`) but omits remaining duplicate keys (`"aaa": "2"`):

```
=> SELECT maptostring (__raw__) FROM darkdata;  
maptostring
```

```
-----  
{  
  "AAA" : "3",  
  "aaa" : "1",  
  "bbb" : "aaa\"bbb"  
}  
(1 row)
```

4. Next, call `maptostring()` with using parameters `canonical_json=false`). This time, the function returns the first duplicate keys and their values:

```
=> SELECT maptostring(__raw__ using parameters canonical_json=false) FROM darkdata;  
maptostring
```

```
-----  
{  
  "aaa": "1",  
  "aaa": "2",  
  "AAA": "3",  
  "bbb": "aaa\"bbb"  
}  
(1 row)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)

- [MAPVALUES](#)
- [MAPVERSION](#)

MAPVALUES

Returns a string representation of the top-level values from a VMap. This transform function requires an `over()` clause.

Syntax

```
mapvalues(VMap_data)
```

Arguments

<code>VMap_data</code>	The VMap from which values should be returned. The VMap can exist as: <ul style="list-style-type: none">• The <code>__raw__</code> column of a flex table• Data returned from a map function such as <code>maplookup()</code>• Other database content
------------------------	---

Examples

The following example shows how to query a `darkmountain` flex table, using an `over()` clause (in this case, the `over(PARTITION BEST)` clause) with `mapvalues()`.

```
=> SELECT * FROM (select mapvalues(darkmountain.__raw__) over(PARTITION BEST) from darkmountain) as  
a;  
  values  
-----  
29029  
34.1  
Everest  
mountain  
29029  
15.4  
Mt St Helens  
volcano  
17000  
12.2  
Denali  
mountain  
14000
```

```
22.8  
Kilimanjaro  
mountain  
50.6  
Mt Washington  
mountain  
(19 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVERSION](#)

MAPVERSION

Returns the version or invalidity of any map data. This scalar function returns the map version (such as 1) or -1, if the map data is invalid.

Syntax

```
mapversion(VMap_data)
```

Arguments

VMap_data	The VMap data either from a <code>__raw__</code> column in a flex table or from the data returned from a map function such as <code>maplookup()</code> .
-----------	--

Examples

The following example shows how to use `mapversion()` with the `darkmountainflex` table, returning `mapversion 1` for the flex table map data:

```
=> SELECT mapversion(__raw__) FROM darkmountain;  
mapversion  
-----  
          1  
          1  
          1  
          1  
          1  
          1  
(5 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSKEY](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
- [MAPTOSTRING](#)
- [MAPVALUES](#)

MATERIALIZED_FLEXTABLE_COLUMNS

Materializes virtual columns listed as *key_names* in the *flextable_keys* table you compute using either [COMPUTE_FLEXTABLE_KEYS](#) or [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#).

Note: Each column that you materialize with this function counts against the data storage limit of your license. To check your Vertica license compliance, call the `AUDIT()` or `AUDIT_FLEX()` functions.

Syntax

```
materialize_flextable_columns('flex_table' [, n-columns [, keys_table_name] ])
```

Arguments

<i>flex_table</i>	<p>The name of the flex table with columns to materialize. Specifying only the flex table name attempts to materialize up to 50 columns of key names in the default <i>flex_table_keys</i> table. When you use this argument, the function:</p> <ul style="list-style-type: none">• Skips any columns already materialized• Ignores any empty keys <p>To materialize a specific number of columns, use the optional parameter <code>n_columns</code>, described next.</p>
<i>n-columns</i>	<p>[Optional] The number of columns to materialize. The function attempts to materialize the number of columns from the <i>flex_table_keys</i> table, skipping any columns already materialized.</p> <p>Vertica tables support a total of 1600 columns, which is the largest value you can specify for <code>n-columns</code>. The function orders the materialized results by frequency, descending, <i>key_name</i> when materializing the first <code>n</code> columns.</p>
<i>keys_table_name</i>	<p>[Optional] The name of a <i>flex_keys_table</i> from which to materialize columns. The function:</p> <ul style="list-style-type: none">• Materializes the number of columns (value of <i>n-columns</i>) from <i>keys_table_name</i>

- Skips any columns already materialized
- Orders the materialized results by frequency, descending, *key_name* when materializing the first n columns.

Examples

The following example shows how to call `materialize_flextable_columns` to materialize columns. First, load a sample file of tweets (`tweets_10000.json`) into the flex table `twitter_r`.

After loading data and computing keys for the sample flex table, call `materialize_flextable_columns` to materialize the first four columns:

```
=> COPY twitter_r FROM '/home/release/KData/tweets_10000.json' parser fjsonparser();
Rows Loaded
-----
      10000
(1 row)

=> SELECT compute_flextable_keys ('twitter_r');
           compute_flextable_keys
-----
Please see public.twitter_r_keys for updated keys
(1 row)

=> select materialize_flextable_columns('twitter_r', 4);
           materialize_flextable_columns
-----
The following columns were added to the table public.twitter_r:
      contributors
      entities.hashtags
      entities.urls
For more details, run the following query:
SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and
table_name = 'twitter_r';

(1 row)
```

The last message in the example recommends querying the `materialize_flextable_columns_results` system table for the results of materializing the columns, as shown:

```
=> SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and
table_name = 'twitter_r';
table_id      | table_schema | table_name |      creation_time      |      key_name      |
status |      message
-----+-----+-----+-----+-----+-----
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.945484-05 | contributors      |
ADDED | Added successfully
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.94551-05 | entities.hashtags |
ADDED | Added successfully
```

```
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.945519-05 | entities.urls |
ADDED | Added successfully
45035996273733172 | public      | twitter_r | 2013-11-20 17:00:27.945532-05 | created_at   |
EXISTS | Column of same name already
(4 rows)
```

See the [MATERIALIZE_FLEXTABLE_COLUMNS_RESULTS](#) system table in the SQL Reference Manual.

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW](#)

RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

Restores the `_keys` table and the `_view`. The function also links the `_keys` table with its associated flex table, in cases where either table is dropped. The function also indicates whether it restored one or both objects.

Syntax

```
RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('flex_table')
```

Arguments

<i>flex_table</i>	The name of a flex table.
-------------------	---------------------------

Examples

This example shows how to invoke this function with an existing flex table, restoring both the `_keys` table and `_view`:

```
=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
       RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW
```

```
-----
The keys table public.darkdata_keys was restored successfully.
```

```
The view public.darkdata_view was restored successfully.
(1 row)
```

This example illustrates that the function restored `darkdata_view`, but that `darkdata_keys` did not need restoring:

```
=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
           RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW
-----
The keys table public.darkdata_keys already exists and is linked to darkdata.
The view public.darkdata_view was restored successfully.
(1 row)
```

After restoring the `_keys` table, there is no content. To populate the flex keys, call the [COMPUTE_FLEXTABLE_KEYS](#) meta function.

```
=> SELECT * FROM darkdata_keys;
 key_name | frequency | data_type_guess
-----+-----+-----
(0 rows)
```

See Also

- [BUILD_FLEXTABLE_VIEW](#)
- [COMPUTE_FLEXTABLE_KEYS](#)
- [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#)
- [MATERIALIZATE_FLEXTABLE_COLUMNS](#)

Hadoop Functions

This section contains functions to manage interactions with Hadoop.

CLEAR_HDFS_CACHES

Clears the configuration information copied from HDFS and any cached connections.

This function affects reads using the `hdfs` scheme in the following ways:

- This function flushes information loaded from configuration files copied from Hadoop (such as `core-site.xml`). These files are found on the path set by the `HadoopConfDir` configuration

parameter.

- This function flushes information about which Name Node is active in a High Availability (HA) Hadoop cluster. Therefore, the first request to Hadoop after calling this function is slower than expected.

Vertica maintains a cache of open connections to name nodes to reduce latency. This function flushes that cache.

Syntax

```
CLEAR_HDFS_CACHES ( )
```

Privileges

Superuser

Example

The following example clears the Hadoop configuration information:

```
=> SELECT CLEAR_HDFS_CACHES();
CLEAR_HDFS_CACHES
-----
Cleared
(1 row)
```

See Also

[Hadoop Parameters](#)

KERBEROS_HDFS_CONFIG_CHECK

Tests the Kerberos configuration of a Vertica cluster that uses HDFS. This function is a more specific version of [KERBEROS_CONFIG_CHECK](#).

You can call this function with arguments to specify an HDFS configuration to test, or without arguments. If you call it with no arguments, this function reads the HDFS configuration files and fails if it does not find them. See [Configuring the hdfs Scheme](#). If it finds configuration files, it tests all configured nameservices.

The function performs the following tests, in order:

- Are Kerberos services available?
- Does a keytab file exist and are the Kerberos and HDFS configuration parameters set in the database?
- Can Vertica read and invoke kinit with the keys to authenticate to HDFS and obtain the database Kerberos ticket?
- Can Vertica perform hdfs and webhdfs operations using both the database Kerberos ticket and user-forwardable tickets?
- Can Vertica make unauthenticated WebHCat calls?

If any test fails, the function returns a descriptive error message.

Syntax

```
KERBEROS_HDFS_CONFIG_CHECK( [ 'hdfsHost:hdfsPort',  
                             'webhdfsHost:webhdfsPort', 'webhcatHost:webhcatPort' ] )
```

Arguments

<i>hdfsHost, hdfsPort</i>	The hostname or IP address and port of the HDFS name node. Vertica uses this server to access data that is specified with hdfs URLs. If the value is '', the function skips this part of the check.
<i>webhdfsHost, webhdfsPort</i>	The hostname or IP address and port of the WebHDFS server. Vertica uses this server to access data that is specified with webhdfs URLs. If the value is '', the function skips this part of the check.
<i>webhcatHost, webhcatPort</i>	The hostname or IP address and port of the WebHCat server. The HCatalog Connector uses this server to access data through Hive. If the value is '', the function skips this part of the check.

Privileges

This function does not require privileges.

Examples

The following example shows the results when the function cannot infer the configuration from the HDFS configuration files.

```
=> SELECT KERBEROS_HDFS_CONFIG_CHECK();
-----Checking basic Kerberos configuration-----
ok: kinit exists
ok: klist exists
ok: krb5 exists at [/etc/krb5.conf]

-----Checking basic Hadoop configuration-----
HadoopFSPrincipal: [mydb/server.example.com@EXAMPLE.COM]
HadoopFSAuthentication (default): [/scratch_b/qa/mydb.keytab]
HadoopFSTokenRefreshFrequency: 0
HadoopFSKinitCommand: export KRB5CCNAME=%tmp%; kinit %principal% -k -t %keytab% -c %tmp%; %cmd%; V_
HADOOP_RESULT=$?; rm %tmp%; exit $V_HADOOP_RESULT;
HadoopFSConnectionTimeout: 60
ok: Can read HDFS keys
ok: Can get tickets for hdfs principal
ok: vertica can kinit as HDFS storage

**Inferring HDFS configuration**
Number of HDFS Clusters: 1
Cluster 1 is HA

-----Checking LibHdfs++ configurations-----

**Checking default LibHdfs++ Nameservice**
ok: Vertica can perform LibHdfs++ operations at [hdfs:///]
ok: Can access [hdfs:///] using delegated ticket for [dbuser]

**Checking LibHdfs++ using hdfs configurations found if any**
ok: Vertica can perform LibHdfs++ operations at [hdfs://ns-server1/]
ok: Can access [hdfs://ns-server1/] using delegated ticket for [dbuser]

-----Checking WebHdfs configurations----- Attempt to pull an unauthenticated page from the
WebHdfs URL via curl
ok: JMX works unauthenticated
ok: Can make external connection to WebHdfs
ok: Vertica can perform WebHdfs operations at [http://server1.example.com:50070/webhdfs/v1/]
ok: Can access [http://server1.example.com:50070/webhdfs/v1/] using delegated ticket for [dbuser]

-----Checking WebHcat configurations-----
ok: Webhcat works unauthenticated
ok: Can connect to webhcat at [server1.example.com:14433]

(1 row)
```

The following example uses parameters to specify the three hosts to check. The Kerberos configuration is valid for HDFS and WebHDFS but not for WebHcat.

```
=> SELECT KERBEROS_HDFS_CONFIG_CHECK('ns-server1',
      'server1.example.com:50070', 'server1.example.com:14433');
```

```
-----Checking basic Kerberos configuration-----
ok: kinit exists
ok: klist exists
ok: krb5 exists at [/etc/krb5.conf]

-----Checking basic Hadoop configuration-----
HadoopFSPrincipal: [mydb/server.example.com@EXAMPLE.COM]
HadoopFSAuthentication (default): [/scratch_b/qa/mydb.keytab]
HadoopFSTokenRefreshFrequency: 0
HadoopFSKinitCommand: export KRB5CCNAME=%tmp%; kinit %principal% -k -t %keytab% -c %tmp%; %cmd%; V_
HADOOP_RESULT=$?; rm %tmp%; exit $V_HADOOP_RESULT;
HadoopFSConnectionTimeout: 60
ok: Can read HDFS keys
ok: Can get tickets for hdfs principal
ok: vertica can kinit as HDFS storage

-----Checking LibHdfs++ configurations-----

**Checking default LibHdfs++ Nameservice**
ok: Vertica can perform LibHdfs++ operations at [hdfs:///]
ok: Can access [hdfs:///] using delegated ticket for [dbuser]

**Checking LibHdfs++ using hdfs configurations found if any**
ok: Vertica can perform LibHdfs++ operations at [hdfs://ns-server1/]
ok: Can access [hdfs://ns-server1/] using delegated ticket for [dbuser]

-----Checking WebHdfs configurations----- Attempt to pull an unauthenticated page from the
WebHdfs URL via curl
ok: JMX works unauthenticated
ok: Can make external connection to WebHdfs
ok: Vertica can perform WebHdfs operations at [http://server1.example.com:50070/webhdfs/v1/]
ok: Can access [http://server1.example.com:50070/webhdfs/v1/] using delegated ticket for [dbuser]

-----Checking WebHcat configurations-----
FAILED: Could not make unauthenticated connection to webhcat
[server1.example.com:14433/templeton/v1]
FAILED: Could not make external connection to webhcat at
[server1.example.com:14433/templeton/v1/status]

(1 row)
```

SYNC_WITH_HCATALOG_SCHEMA

Copies the structure of a Hive database schema available through the HCatalog Connector to a Vertica schema.

Syntax

```
SYNC_WITH_HCATALOG_SCHEMA( local_schema, hcatalog_schema, [drop_tables] )
```

Parameters

<i>Local_schema</i>	The existing Vertica schema to store the copied HCatalog schema's metadata. This can be the same schema as <i>hcatalog_schema</i> , or it can be a separate one created using CREATE SCHEMA.
<i>hcatalog_schema</i>	The HCatalog schema to copy. This is the schema created using CREATE HCATALOG SCHEMA.
[<i>drop_tables</i>]	If true, drop any tables in <i>Local_schema</i> that do not correspond to a table in <i>hcatalog_schema</i>

Notes

The SYNC_WITH_HCATALOG_SCHEMA function overwrites tables in the Vertica schema whose names match a table in the HCatalog schema. Do not use the Vertica schema to store other data.

Hive STRING and BINARY data types are matched, in Vertica, to VARCHAR(65000) and VARBINARY(65000) types. You can use [ALTER TABLE](#) to adjust these after creating the schema. The maximum size of a VARCHAR or VARBINARY in Vertica is 65000, but you can use LONG VARCHAR and LONG VARBINARY to specify larger values.

Hive and Vertica define string length in different ways. In Hive the length is the number of characters; in Vertica it is the number of bytes. Thus, a character encoding that uses more than one byte, such as Unicode, can result in mismatches between the two. Set values in Vertica based on bytes, not characters, to avoid data truncation.

If the data size exceeds the declared size for the column, Vertica logs an event at read time in the QUERY_EVENTS system table.

This function can synchronize the HCatalog schema directly, in which case you call it with the same schema name for both parameters. It can also synchronize a different schema to the HCatalog schema.

If you change the settings of any HCatalog Connector configuration parameters ([Hadoop Parameters](#)), you must call this function again.

Privileges

The user must have CREATE privileges on *Local_schema* and USAGE permissions on *hcatalog_schema*.

Examples

The following example shows using `SYNC_WITH_HCATALOG_SCHEMA` to synchronize an HCatalog schema named `hcat`:

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcahost' HCATALOG_SCHEMA='default'
-> HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat', 'hcat');
sync_with_hcatalog_schema
-----
Schema hcat synchronized with hcat
tables in hcat = 56
tables altered in hcat = 0
tables created in hcat = 56
stale tables in hcat = 0
table changes erred in hcat = 0
(1 row)

=> -- Use vsql's \d command to describe a table in the synced schema

=> \d hcat.messages
List of Fields by Tables
 Schema | Table | Column | Type | Size | Default | Not Null | Primary Key | Foreign
Key
-----+-----+-----+-----+-----+-----+-----+-----+-----
-----
hcat    | messages | id      | int   | 8    |         | f        | f           |
hcat    | messages | userid  | varchar(65000) | 65000 |         | f        | f           |
hcat    | messages | "time"  | varchar(65000) | 65000 |         | f        | f           |
hcat    | messages | message | varchar(65000) | 65000 |         | f        | f           |
(4 rows)
```

The following example shows using `SYNC_WITH_HCATALOG_SCHEMA` followed by `ALTER TABLE` to adjust a column value:

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcahost' HCATALOG_SCHEMA='default'
-> HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat', 'hcat');
...
=> ALTER TABLE hcat.t ALTER COLUMN a1 SET DATA TYPE long varchar(1000000);
=> ALTER TABLE hcat.t ALTER COLUMN a2 SET DATA TYPE long varbinary(1000000);
```

The following example shows using `SYNC_WITH_HCATALOG_SCHEMA` with a local (non-HCatalog) schema:

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcahost' HCATALOG_SCHEMA='default'
-> HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> CREATE SCHEMA hcat_local;
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat_local', 'hcat');
```

SYNC_WITH_HCATALOG_SCHEMA_TABLE

Copies the structure of a single table in a Hive database schema available through the HCatalog Connector to a Vertica table.

Syntax

```
SYNC_WITH_HCATALOG_SCHEMA_TABLE( local_schema, hcatalog_schema, table_name )
```

Parameters

<i>local_schema</i>	The existing Vertica schema to store the copied HCatalog schema's metadata. This can be the same schema as <i>hcatalog_schema</i> , or it can be a separate one created using CREATE SCHEMA.
<i>hcatalog_schema</i>	The HCatalog schema to copy. This is the schema created using CREATE HCATALOG SCHEMA.
<i>table_name</i>	The table in <i>hcatalog_schema</i> to copy.

Notes

If *table_name* does not exist in *hcatalog_schema*, this function returns an error.

If *table_name* already exists in *local_schema*, the SYNC_WITH_HCATALOG_SCHEMA_TABLE function overwrites it.

Hive STRING and BINARY data types are matched, in Vertica, to VARCHAR(65000) and VARBINARY(65000) types. You can use [ALTER TABLE](#) to adjust these after creating the schema. The maximum size of a VARCHAR or VARBINARY in Vertica is 65000, but you can use LONG VARCHAR and LONG VARBINARY to specify larger values.

Hive and Vertica define string length in different ways. In Hive the length is the number of characters; in Vertica it is the number of bytes. Thus, a character encoding that uses more than one byte, such as Unicode, can result in mismatches between the two. Set values in Vertica based on bytes, not characters, to avoid data truncation.

If the data size exceeds the declared size for the column, Vertica logs an event at read time in the QUERY_EVENTS system table.

This function can synchronize the HCatalog schema directly, in which case you call it with the same schema name for both of the *local_schema* and *hcatalog_schema* parameters. It can also synchronize a different schema to the HCatalog schema.

Privileges

The user must have CREATE privileges on *Local_schema* and USAGE permissions on *hcatalog_schema*.

Examples

The following example shows using SYNC_WITH_HCATALOG_SCHEMA_TABLE to synchronize the "nation" table:

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcatost' HCATALOG_SCHEMA='hcat_truth'  
-> HCATALOG_USER='hcatuser';  
CREATE SCHEMA  
=> SELECT sync_with_hcatalog_schema_table('hcat', 'hcat_truth', 'nation');  
sync_with_hcatalog_schema_table  
-----  
Schema hcat synchronized with hcat_truth for table nation  
table nation is created in schema hcat  
(1 row)
```

The following example shows the behavior if the "nation" table already exists in the local schema:

```
=> SELECT sync_with_hcatalog_schema_table('hcat', 'hcat_truth', 'nation');  
sync_with_hcatalog_schema_table  
-----  
Schema hcat synchronized with hcat_truth for table nation  
table nation is altered in schema hcat  
(1 row)
```

VERIFY_HADOOP_CONF_DIR

Verifies that the Hadoop configuration that is used to access HDFS is valid on all Vertica nodes. The configuration is valid if:

- all required configuration files are found on the path defined by the HadoopConfDir configuration parameter
- all properties needed by Vertica are set in those files

This function does not attempt to validate the settings of those properties; it only verifies that they have values.

It is possible for Hadoop configuration to be valid on some nodes and invalid on others. The function reports a validation failure if the value is invalid on any node; the rest of the output reports the details.

Syntax

```
VERIFY_HADOOP_CONF_DIR( )
```

Parameters

This function has no parameters.

Privileges

This function does not require privileges.

Examples

The following example shows the results when the Hadoop configuration is valid.

```
=> SELECT VERIFY_HADOOP_CONF_DIR();
       verify_hadoop_conf_dir
-----
Validation Success
v_vmart_node0001: HadoopConfDir [PG_TESTOUT/config] is valid
v_vmart_node0002: HadoopConfDir [PG_TESTOUT/config] is valid
v_vmart_node0003: HadoopConfDir [PG_TESTOUT/config] is valid
v_vmart_node0004: HadoopConfDir [PG_TESTOUT/config] is valid
(1 row)
```

In the following example, the Hadoop configuration is valid on one node, but on other nodes a needed value is missing.

```
=> SELECT VERIFY_HADOOP_CONF_DIR();
       verify_hadoop_conf_dir
-----
Validation Failure
v_vmart_node0001: HadoopConfDir [PG_TESTOUT/test_configs/config] is valid
v_vmart_node0002: No fs.defaultFS parameter found in config files in [PG_TESTOUT/config]
v_vmart_node0003: No fs.defaultFS parameter found in config files in [PG_TESTOUT/config]
v_vmart_node0004: No fs.defaultFS parameter found in config files in [PG_TESTOUT/config]
(1 row)
```

LDAP Link Functions

This section contains the functions associated with the Vertica [LDAP Link](#) function.

In This Section

LDAP_LINK_SYNC_START

Begins the synchronization between the LDAP server and Vertica immediately rather than waiting for the interval set in LDAPLinkInterval.

Syntax

```
ldap_link_sync_start()
```

Privileges

You must be a dbadmin user.

Example

```
=> SELECT ldap_link_sync_start();
```

See Also

[LDAP Link Parameters](#)

License Management Functions

This section contains function that monitor Vertica license status and compliance.

AUDIT

Estimates the raw data size of a database, schema, or table as it is counted in an audit of the database size.

AUDIT estimates the size using the same data sampling method as Vertica uses, to determine if a database complies with the licensed database size allowance. The results of this function are not considered when Vertica determines whether the size of the database complies with the Vertica license's data allowance. For details, see [Calculating the Database Size](#) in the Administrator's Guide.

Syntax

```
AUDIT('name'[, 'granularity'] [, error-tolerance[, confidence-level] ])
```

Parameters

<i>name</i>	The database, schema, or table to audit. If you supply an empty string (' '), AUDIT audits the database.
<i>granularity</i>	<p>The level at which the audit reports its results, one of the following strings:</p> <ul style="list-style-type: none">• database• schema• table <p>The level of granularity must be equal to or less than the granularity of <i>name</i>. If you omit this parameter, granularity is the same level as <i>name</i>. Thus, if <code>online_sales</code> is a schema, the following statements are identical:</p> <pre>AUDIT('sales', 'schema'); AUDIT('sales');</pre>

	<p>If granularity is less than <i>name</i>, AUDIT returns with a message that refers to system table <code>V_CATALOG.USER_AUDITS</code>. You can query this table to find detailed information on the objects within <i>name</i> at the specified granularity. For example:</p> <pre> => SELECT AUDIT('online_sales','table'); AUDIT ----- See table sizes in v_catalog.user_audits for schema online_sales (1 row) => SELECT * FROM user_audits WHERE object_schema='online_sales'; -[RECORD 1]-----+----- size_bytes 59016 user_id 45035996273704962 user_name dbadmin object_id 45035996273739208 object_type TABLE object_schema online_sales object_name online_page_dimension license_name vertica audit_start_timestamp 2016-06-17 10:37:31.238291-04 audit_end_timestamp 2016-06-17 10:37:31.358232-04 confidence_level_percent 99 error_tolerance_percent 20 used_sampling f confidence_interval_lower_bound_bytes 59016 confidence_interval_upper_bound_bytes 59016 sample_count 0 cell_count 0 -[RECORD 2]-----+----- ... </pre>
<p><i>error-tolerance</i></p>	<p>Specifies the percentage margin of error allowed in the audit estimate. Enter the tolerance value as a decimal number, between 0 and 100. The default value is 5, for a 5% margin of error.</p> <p>Setting this value to 0 results in a full database audit, which is very resource intensive, as AUDIT analyzes the entire database. A full database audit significantly impacts performance, so Vertica does not recommend it for a production database</p> <p>Due to the iterative sampling that the auditing process uses, making the error tolerance a small fraction of a percent (for example, 0.00001) can cause AUDIT to run for a longer period than a full database audit. The lower you specify this value, the more resources the audit uses, as it performs more data sampling.</p>
<p><i>confidence-level</i></p>	<p>Specifies the statistical confidence level percentage of the estimate. Enter the confidence value as a decimal number, between 0 and 100. The default value is 99, indicating a confidence level of 99%.</p>

The higher the confidence value, the more resources the function uses, since it performs more data sampling. Setting this value to 100 results in a full audit of the database, which is very resource intensive, as the function analyzes all of the database. A full database audit significantly impacts performance, so Vertica does not recommend it for a production database.

Privileges

Non-superuser:

- Table: SELECT privilege
- Schema: USAGE privilege
- Database: Only audits the size of data that the user has permission to access

Examples

Audit the entire database:

```
=> SELECT AUDIT('');  
AUDIT  
-----  
76376696  
(1 row)
```

Audit the database with 25% error tolerance:

```
=> SELECT AUDIT('',25);  
AUDIT  
-----  
75797126  
(1 row)
```

Audit the database with a 25% level of tolerance and a 0% confidence level:

```
=> SELECT AUDIT('',25,90);  
AUDIT  
-----  
76402672  
(1 row)
```

Audit schema `online_sales`:

```
=> SELECT AUDIT('online_sales');  
AUDIT
```

```
-----
35716504
(1 row)
```

Audit schema `online_sales` and report the results by table:

```
=> SELECT AUDIT('online_sales','table');
          AUDIT
-----
See table sizes in v_catalog.user_audits for schema
online_sales
(1 row)
=> \x
Expanded display is on.
=> SELECT * FROM user_audits WHERE object_schema =
'online_sales';
-[ RECORD 1 ]-----+-----
size_bytes          | 64960
user_id             | 45035996273704962
user_name           | dbadmin
object_id           | 45035996273717636
object_type         | TABLE
object_schema       | online_sales
object_name         | online_page_dimension
audit_start_timestamp | 2011-04-05 09:24:48.224081-04
audit_end_timestamp | 2011-04-05 09:24:48.337551-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling       | f
confidence_interval_lower_bound_bytes | 64960
confidence_interval_upper_bound_bytes | 64960
sample_count        | 0
cell_count          | 0
-[ RECORD 2 ]-----+-----
size_bytes          | 20197
user_id             | 45035996273704962
user_name           | dbadmin
object_id           | 45035996273717640
object_type         | TABLE
object_schema       | online_sales
object_name         | call_center_dimension
audit_start_timestamp | 2011-04-05 09:24:48.340206-04
audit_end_timestamp | 2011-04-05 09:24:48.365915-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling       | f
confidence_interval_lower_bound_bytes | 20197
confidence_interval_upper_bound_bytes | 20197
sample_count        | 0
cell_count          | 0
-[ RECORD 3 ]-----+-----
size_bytes          | 35614800
user_id             | 45035996273704962
user_name           | dbadmin
object_id           | 45035996273717644
object_type         | TABLE
object_schema       | online_sales
object_name         | online_sales_fact
```

```
audit_start_timestamp | 2011-04-05 09:24:48.368575-04
audit_end_timestamp   | 2011-04-05 09:24:48.379307-04
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling         | t
confidence_interval_lower_bound_bytes | 34692956
confidence_interval_upper_bound_bytes | 36536644
sample_count          | 10000
cell_count            | 9000000
```

AUDIT_FLEX

Estimates the ROS size of one or more flexible tables contained in a database, schema, or projection.

The `audit_flex()` function measures the export size of the flex data values (does not include flex keys) for the `__raw__` column of one or more flexible tables. The function does not audit other flex table columns that are created as, or promoted to, real columns. Temporary flex tables are not included in the audit.

Each time a user calls `audit_flex()`, Vertica stores the results in the `V_CATALOG.USER_AUDITS` system table.

Syntax

```
AUDIT_FLEX (name)
```

Parameters

<i>name</i>	<p>Specifies what database entity to audit. Enter the entity name as a string in single quotes (' '), as follows:</p> <ul style="list-style-type: none">• Empty string (' ') — Return the size of the ROS containers for all flexible tables in the database. You cannot enter the database name.• Schema name (' schema_name ') — Return the size of all <code>__raw__</code> columns of flexible tables in <code>schema_name</code>.• A projection name (' proj_name ') — Return the ROS size of a projection for a <code>__raw__</code> column.• A flex table name (' flex_table_name ') — Return the ROS size of a flex table's <code>__raw__</code> column.
-------------	---

Privileges

- SELECT privilege on table
- USAGE privilege on schema

Note: AUDIT_FLEX() works only on the flexible tables, projections, schemas, and databases to which the user has permissions.

Examples

To audit the flex tables in the database:

```
dbs=> select audit_flex('');
audit_flex
-----
8567679
(1 row)
```

To audit the flex tables in a specific schema, such as public:

```
dbs=> select audit_flex('public');
audit_flex
-----
8567679
(1 row)
```

To audit the flex tables in a specific projection, such as bakery_b0:

```
dbs=> select audit_flex('bakery_b0');
audit_flex
-----
8566723
(1 row)
```

To audit a flex table, such as bakery:

```
dbs=> select audit_flex('bakery');
audit_flex
-----
8566723
(1 row)
```

To report the results of all audits saved in the USER_AUDITS, the following shows part of an extended display from the system table showing an audit run on a schema called test, and the entire database, dbs:

```
db>=> \x
Expanded display is on.

db>=> select * from user_audits;
-[ RECORD 1 ]-----+-----
size_bytes          | 0
user_id             | 45035996273704962
user_name           | release
object_id           | 45035996273736664
object_type         | SCHEMA
object_schema       |
object_name         | test
audit_start_timestamp | 2014-02-04 14:52:15.126592-05
audit_end_timestamp  | 2014-02-04 14:52:15.139475-05
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling       | f
confidence_interval_lower_bound_bytes | 0
confidence_interval_upper_bound_bytes | 0
sample_count        | 0
cell_count          | 0
-[ RECORD 2 ]-----+-----
size_bytes          | 38051
user_id             | 45035996273704962
user_name           | release
object_id           | 45035996273704974
object_type         | DATABASE
object_schema       |
object_name         | dbs
audit_start_timestamp | 2014-02-05 13:44:41.11926-05
audit_end_timestamp  | 2014-02-05 13:44:41.227035-05
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling       | f
confidence_interval_lower_bound_bytes | 38051
confidence_interval_upper_bound_bytes | 38051
sample_count        | 0
cell_count          | 0
-[ RECORD 3 ]-----+-----
.
.
.
```

AUDIT_LICENSE_SIZE

Triggers an immediate audit of the database size to determine if it is in compliance with the raw data storage allowance included in your Vertica licenses.

Syntax

```
AUDIT_LICENSE_SIZE()
```

Privileges

Superuser

Example

```
=> SELECT audit_license_size();
  audit_license_size
-----
Raw Data Size: 0.00TB +/- 0.00TB
License Size : 10.00TB
Utilization  : 0%
Audit Time   : 2015-09-24 12:19:15.425486-04
Compliance Status : The database is in compliance with respect to raw data size.

License End Date: 2015-11-23 00:00:00 Days Remaining: 60.53
(1 row)
```

AUDIT_LICENSE_TERM

Triggers an immediate audit to determine if the Vertica license has expired.

Syntax

```
AUDIT_LICENSE_TERM()
```

Privileges

Superuser

Example

```
=> SELECT audit_license_term();
  audit_license_term
-----
Raw Data Size: 0.00TB +/- 0.00TB
License Size : 10.00TB
Utilization  : 0%
Audit Time   : 2015-09-24 12:19:15.425486-04
Compliance Status : The database is in compliance with respect to raw data size.

License End Date: 2015-11-23 00:00:00 Days Remaining: 60.53
(1 row)
```

GET_AUDIT_TIME

Reports the time when the automatic audit of database size occurs. Vertica performs this audit if your Vertica license includes a data size allowance. For details of this audit, see [Managing Licenses](#) in the Administrator's Guide. To change the time the audit runs, use the [SET_AUDIT_TIME](#) function.

Syntax

```
GET_AUDIT_TIME()
```

Privileges

None

Example

```
=> SELECT get_audit_time();
get_audit_time
-----
The audit is scheduled to run at 11:59 PM each day.
(1 row)
```

GET_COMPLIANCE_STATUS

Displays whether your database is in compliance with your Vertica license agreement. This information includes the results of Vertica's most recent audit of the database size (if your license has a data allowance as part of its terms), and the license term (if your license has an end date).

GET_COMPLIANCE_STATUS measures data allowance by TBs (where a TB equals 1024⁴ bytes).

The information displayed by GET_COMPLIANCE_STATUS includes:

- The estimated size of the database (see [How Vertica Calculates Database Size](#) in the Administrator's Guide for an explanation of the size estimate).
- The raw data size allowed by your Vertica license.
- The percentage of your allowance that your database is currently using.

- The date and time of the last audit.
- Whether your database complies with the data allowance terms of your license agreement.
- The end date of your license.
- How many days remain until your license expires.

Note: If your license does not have a data allowance or end date, some of the values may not appear in the output for `GET_COMPLIANCE_STATUS`.

If the audit shows your license is not in compliance with your data allowance, you should either delete data to bring the size of the database under the licensed amount, or upgrade your license. If your license term has expired, you should contact Micro Focus immediately to renew your license. See [Managing Licenses](#) in the Administrator's Guide for further details.

Syntax

```
GET_COMPLIANCE_STATUS()
```

Privileges

None

Example

```
=> SELECT get_compliance_status();
  get_compliance_status
-----
Raw Data Size: 0.00TB +/- 0.00TB
License Size : 10.00TB
Utilization  : 0%
Audit Time   : 2015-09-24 12:19:15.425486-04
Compliance Status : The database is in compliance with respect to raw data size.

License End Date: 2015-11-23 00:00:00 Days Remaining: 60.53
(1 row)
```

DISPLAY_LICENSE

Returns the terms of your Vertica license. The information this function displays is:

- The start and end dates for which the license is valid (or "Perpetual" if the license has no expiration).
- The number of days you are allowed to use Vertica after your license term expires (the grace period)
- The amount of data your database can store, if your license includes a data allowance.

Syntax

```
DISPLAY_LICENSE()
```

Privileges

None

Examples

```
=> SELECT DISPLAY_LICENSE();
          DISPLAY_LICENSE
-----
Vertica Systems, Inc.
1/1/2011
12/31/2011
30
50TB
(1 row)
```

SET_AUDIT_TIME

Sets the time that Vertica performs automatic database size audit to determine if the size of the database is compliant with the raw data allowance in your Vertica license. Use this function if the audits are currently scheduled to occur during your database's peak activity time. This is normally not a concern, since the automatic audit has little impact on database performance.

Audits are scheduled by the preceding audit, so changing the audit time does not affect the next scheduled audit. For example, if your next audit is scheduled to take place at 11:59PM and you use `SET_AUDIT_TIME` to change the audit schedule 3AM, the previously scheduled 11:59PM audit still runs. As that audit finishes, it schedules the next audit to occur at 3AM.

Vertica always performs the next scheduled audit even where you have changed the audit time using `SET_AUDIT_TIME` and then triggered an automatic audit by issuing the statement, `SELECT AUDIT_LICENSE_SIZE`. Only after the next scheduled audit does Vertica begin auditing at the new time you set using `SET_AUDIT_TIME`. Thereafter, Vertica audits at the new time.

Syntax

`SET_AUDIT_TIME(time)`

<i>time</i>	A string containing the time in 'HH:MM AM/PM' format (for example, '1:00 AM') when the audit should run daily.
-------------	--

Privileges

Superuser

Example

```
=> SELECT SET_AUDIT_TIME('3:00 AM');
          SET_AUDIT_TIME
-----
The scheduled audit time will be set to 3:00 AM after the next audit.
(1 row)
```

Multiple Active Result Sets Functions

This section contains the functions associated with the Vertica library for Multiple Active Result Sets (MARS).

CLOSE_ALL_RESULTSETS

Closes all result set sessions within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

Syntax

```
SELECT CLOSE_ALL_RESULTSETS ('session_id')
```

Parameters

<i>session_id</i>	A string that specifies the Multiple Active Result Sets session.
-------------------	--

Privileges

None; however, without superuser privileges, you can only close your own session's results.

Examples

This example shows how you can view a MARS result set, then close the result set, and then confirm that the result set has been closed.

Query the MARS storage table. One session ID is open and three result sets appear in the output.

```
SELECT * FROM SESSION_MARS_STORE;
```

node_name	session_id	user_name	resultset_id	row_count	remaining_row_count	bytes_used
v_vmart_node0001	server1.company.-83046:1y28gu9	dbadmin	7	777460	776460	89692848
v_vmart_node0001	server1.company.-83046:1y28gu9	dbadmin	8	324349	323349	81862010
v_vmart_node0001	server1.company.-83046:1y28gu9	dbadmin	9	277947	276947	32978280

(1 row)

Close all result sets for session server1.company.-83046:1y28gu9:

```
SELECT CLOSE_ALL_RESULTSETS('server1.company.-83046:1y28gu9');
```

close_all_resultsets

Closing all result sets from server1.company.-83046:1y28gu9

(1 row)

Query the MARS storage table again for the current status. You can see that the session and result sets have been closed:

```
SELECT * FROM SESSION_MARS_STORE;
```

node_name	session_id	user_name	resultset_id	row_count	remaining_row_count	bytes_used
-----------	------------	-----------	--------------	-----------	---------------------	------------

(0 rows)

CLOSE_RESULTSET

Closes a specific result set within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

Syntax

```
SELECT CLOSE_RESULTSET ('session_id', ResultSetID)
```

Parameters

session_id	A string that specifies the Multiple Active Result Sets session containing the ResultSetID to close.
ResultSetID	An integer that specifies which result set to close.

Privileges

None; however, without superuser privileges, you can only close your own session's results.

Examples

This example shows a MARS storage table opened. One session_id is currently open, and one result set appears in the output.

```
SELECT * FROM SESSION_MARS_STORE;
   node_name      | session_id          | user_name | resultset_id | row_count |
remaining_row_count | bytes_used
-----+-----+-----+-----+-----+-----
v_vmart_node0001 | server1.company.-83046:1y28gu9 | dbadmin  |             1 |      318718 |
      312718 |      80441904
(1 row)
```

Close user session server1.company.-83046:1y28gu9 and result set 1:

```
SELECT CLOSE_RESULTSET('server1.company.-83046:1y28gu9', 1);
      close_resultset
-----
Closing result set 1 from server1.company.-83046:1y28gu9
(1 row)
```

Query the MARS storage table again for current status. You can see that result set 1 is now closed:

```
SELECT * FROM SESSION_MARS_STORE;
```

```
node_name      | session_id      | user_name | resultset_id | row_count |
remaining_row_count | bytes_used
-----+-----+-----+-----+-----+-----
(0 rows)
```

Partition Management Functions

This section contains partition management functions specific to Vertica.

COPY_PARTITIONS_TO_TABLE

Copies partitions from one table to another. This lightweight partition copy increases performance by initially sharing the same storage between two tables. After the copy operation is complete, the tables are independent of each other. Users can perform operations on one table without impacting the other. These operations can increase the overall storage required for both tables.

Note: Although they share storage space, Vertica considers the partitions as discrete objects for license capacity purposes. For example, copying a one TB partition would only consume one TB of space. Your Vertica license, however, considers them as separate objects consuming two TB of space.

Syntax

```
COPY_PARTITIONS_TO_TABLE (  
    '[schema.]source-table',  
    'min-range-value',  
    'max-range-value',  
    '[schema.]target-table'  
)
```

Parameters

<i>schema</i>	Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>source-table</i>	The source table of the partitions to copy.
<i>min-range-value</i>	The minimum value of the partition to copy. To copy one partition, the minimum and maximum values must be the same.
<i>max-range-value</i>	The maximum value of the partition to be copied. To copy one partition, the minimum and maximum values must be the same.

<i>target-table</i>	The target table of the partitions to copy. If the table does not exist, Vertica creates a table from the source table's definition, by calling CREATE TABLE with LIKE and INCLUDING PROJECTIONS clause. The new table inherits ownership from the source table. For details, see Replicating a Table ..
---------------------	--

Privileges

- Ownership or USAGE privileges on the source table.
- CREATE privileges on the target table, if COPY_PARTITIONS_TO_TABLE creates it.

Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partitioning expression
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled. For more information on constraints, refer to [Enforcing Primary Key and Foreign Key Constraints](#).

Note: If the target table has primary or unique key constraints enabled and copying or moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation.

- Check constraints. For [MOVE_PARTITIONS_TO_TABLE](#) and [COPY_PARTITIONS_TO_TABLE](#), Vertica enforces enabled check constraints on the target table only. For [SWAP_PARTITIONS_BETWEEN_TABLES](#), Vertica enforces enabled check constraints on both tables. If there is a violation of an enabled check constraint, Vertica rolls back the operation.
- Number and definitions of text indices.

Table Restrictions

The following restrictions apply to the source and target tables:

- If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.
- The following tables cannot be used as sources or targets:
 - Temporary tables
 - Virtual tables
 - System tables
 - External tables

Examples

If you call `COPY_PARTITIONS_TO_TABLE` and the target table does not exist, the function creates the table automatically. In the following example, the target table `partn_backup.tradfes_200801` does not exist. `COPY_PARTITIONS_TO_TABLE` creates the table and replicates the partition. Vertica also copies all the constraints associated with the source table except foreign key constraints.

```
=> SELECT COPY_PARTITIONS_TO_TABLE (  
        'prod_trades',  
        '200801',  
        '200801',  
        'partn_backup.tradfes_200801');  
COPY_PARTITIONS_TO_TABLE  
-----  
 1 distinct partition values copied at epoch 15.  
(1 row)
```

See Also

[Archiving Partitions](#)

DROP_PARTITION

Drops the specified partition. If the WOS contains table data, `DROP_PARTITION` forces a moveout before it executes the drop operation.

Syntax

```
DROP_PARTITION (
  '[schema.]table-name',
  partition-value
  [, ignore-moveout-errors]
  [, reorganize-data ]
)
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	The target table. The table cannot be used as a dimension table in a pre-join projection and cannot have out-of-date (unrefreshed) projections.
<i>partition-value</i>	The key of the partition to drop.
<i>ignore-moveout-errors</i>	<p>Optional Boolean argument:</p> <ul style="list-style-type: none"> <code>true</code>: Ignores any WOS moveout errors and forces the operation to continue. Set this parameter to <code>true</code> only if there is no WOS data for the partition. <code>false</code> (default): Displays any moveout errors and aborts the operation on error. <p>Note: If you set this parameter to <code>true</code> and the WOS includes data for the partition in WOS, partition data in WOS is not dropped.</p>
<i>reorganize-data</i>	<p>Optional Boolean argument:</p> <ul style="list-style-type: none"> <code>true</code>: Reorganizes the data if it is not organized, and then drops the partition. <code>false</code>: Does not attempt to reorganize the data before dropping the partition. If this parameter is <code>false</code> and the function encounters a ROS without partition keys, an

	error occurs.
--	---------------

Privileges

One of the following:

- DBADMIN
- Table owner
- USAGE privileges on the table schema and TRUNCATE privileges on the table

Examples

See [Dropping Partitions](#) in the Administrator's Guide.

See Also

- [DUMP_PARTITION_KEYS](#)
- [DUMP_PROJECTION_PARTITION_KEYS](#)
- [DUMP_TABLE_PARTITION_KEYS](#)
- [PARTITION_TABLE](#)

DUMP_PROJECTION_PARTITION_KEYS

Dumps the partition keys of the specified projection.

Syntax

```
DUMP_PROJECTION_PARTITION_KEYS( '[schema.]projection-name'
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
---------------	---

<i>projection-name</i>	The projection name.
------------------------	----------------------

Privileges

- SELECT privilege on table
- USAGE privileges on schema

Example

The following statements create the table and projection `online_sales.online_sales_fact` and `online_sales.online_sales_fact_unseg_p`, respectively, and partitions table data by the column `call_center_key`:

```
=> CREATE TABLE online_sales.online_sales_fact
(
  sale_date_key int NOT NULL,
  ship_date_key int NOT NULL,
  product_key int NOT NULL,
  product_version int NOT NULL,
  customer_key int NOT NULL,
  call_center_key int NOT NULL,
  online_page_key int NOT NULL,
  shipping_key int NOT NULL,
  warehouse_key int NOT NULL,
  promotion_key int NOT NULL,
  pos_transaction_number int NOT NULL,
  sales_quantity int,
  sales_dollar_amount float,
  ship_dollar_amount float,
  net_dollar_amount float,
  cost_dollar_amount float,
  gross_profit_dollar_amount float,
  transaction_type varchar(16)
)
PARTITION BY (online_sales_fact.call_center_key);

=> CREATE PROJECTION online_sales.online_sales_fact_unseg_p AS SELECT * from online_sales.online_sales_fact unsegmented all nodes;
```

The following `DUMP_PROJECTION_PARTITION_KEYS` statement dumps the partition key from the projection `online_sales.online_sales_fact_unseg_p`:

```
=> SELECT DUMP_PROJECTION_PARTITION_KEYS('online_sales.online_sales_fact_unseg_p');

Partition keys on node v_vmart_node0001
Projection 'online_sales_fact_unseg_p'
Storage [ROS container]
  No of partition keys: 1
  Partition keys: 200
Storage [ROS container]
```

```
No of partition keys: 1
Partition keys: 199
...
Storage [ROS container]
  No of partition keys: 1
  Partition keys: 2
Storage [ROS container]
  No of partition keys: 1
  Partition keys: 1

Partition keys on node v_vmart_node0002
Projection 'online_sales_fact_unseg_p'
Storage [ROS container]
  No of partition keys: 1
  Partition keys: 200
Storage [ROS container]
  No of partition keys: 1
  Partition keys: 199
...
(1 row)
```

See Also

- [Using Table Partitions](#)
- [DO_TM_TASK](#)
- [DROP_PARTITION](#)
- [DUMP_PARTITION_KEYS](#)
- [DUMP_TABLE_PARTITION_KEYS](#)
- [PARTITION_PROJECTION](#)
- [PARTITION_TABLE](#)
- [PROJECTIONS](#)

DUMP_TABLE_PARTITION_KEYS

Dumps the partition keys of all projections for the specified table.

Syntax

```
DUMP_TABLE_PARTITION_KEYS ( 'table-name' )
```

Parameters

<i>table-name</i>	The name of the table.
-------------------	------------------------

Privilege

- SELECT privilege on table
- USAGE privileges on schema

Examples

The following example creates a simple table called `states` and partitions the data by state:

```
=> CREATE TABLE states (year INTEGER NOT NULL,  
    state VARCHAR NOT NULL)  
    PARTITION BY state;  
=> CREATE PROJECTION states_p (state, year) AS  
    SELECT * FROM states  
    ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now dump the partition keys of all projections anchored on table `states`:

```
=> SELECT DUMP_TABLE_PARTITION_KEYS( 'states' );  
    DUMP_TABLE_PARTITION_KEYS  
-----  
Partition keys on node v_vmart_node0001  
Projection 'states_p'  
Storage [ROS container]  
  No of partition keys: 1  
  Partition keys: VT  
Storage [ROS container]  
  No of partition keys: 1  
  Partition keys: PA  
Storage [ROS container]  
  No of partition keys: 1  
  Partition keys: NY  
Storage [ROS container]  
  No of partition keys: 1  
  Partition keys: MA  
  
Partition keys on node v_vmart_node0002  
...  
(1 row)
```

See Also

- [DO_TM_TASK](#)
- [DROP_PARTITION](#)
- [DUMP_PROJECTION_PARTITION_KEYS](#)
- [DUMP_TABLE_PARTITION_KEYS](#)
- [PARTITION_PROJECTION](#)
- [PARTITION_TABLE](#)
- [Using Table Partitions](#)

MOVE_PARTITIONS_TO_TABLE

Moves partitions from one table to another.

Syntax

```
MOVE_PARTITIONS_TO_TABLE (  
  '[schema.]source-table',  
  'min-range-value',  
  'max-range-value',  
  '[schema.]target-table'  
)
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>source-table</i>	The source table of the partitions to move.
<i>min-range-value</i>	The minimum value of the partition to move. To move one partition, the minimum and maximum values must be the same.
<i>max-range-value</i>	The maximum value of the partition to be moved. To move one partition, the minimum and maximum values must be the same.

<i>target-table</i>	The target table of the partitions to move. If the table does not exist, Vertica creates a table from the source table's definition, by calling <code>CREATE TABLE</code> with <code>LIKE</code> and <code>INCLUDING PROJECTIONS</code> clause. The new table inherits ownership from the source table. For details, see Replicating a Table .
---------------------	--

Privileges

If the target table does not exist, you must have `CREATE` privileges on the target schema, to enable table creation. One of the following conditions is also required:

- `DBADMIN` role
- Owner of the source and target tables
- `USAGE` privileges on source and target schemas, `TRUNCATE` privileges on the source table, and `INSERT` privileges on the target table

Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including `NULL/NOT NULL` constraints
- Segmentation
- Partitioning expression
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled. For more information on constraints, refer to [Enforcing Primary Key and Foreign Key Constraints](#).

Note: If the target table has primary or unique key constraints enabled and copying or moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation.

- Check constraints. For `MOVE_PARTITIONS_TO_TABLE` and `COPY_PARTITIONS_TO_TABLE`, Vertica enforces enabled check constraints on the target table only. For `SWAP_`

[PARTITIONS_BETWEEN_TABLES](#), Vertica enforces enabled check constraints on both tables. If there is a violation of an enabled check constraint, Vertica rolls back the operation.

- Number and definitions of text indices.

Table Restrictions

The following restrictions apply to the source and target tables:

- If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.
- The following tables cannot be used as sources or targets:
 - Temporary tables
 - Virtual tables
 - System tables
 - External tables

Examples

See [Archiving Partitions](#).

See Also

- [COPY_PARTITIONS_TO_TABLE](#)
- [SWAP_PARTITIONS_BETWEEN_TABLES](#)

PARTITION_PROJECTION

Splits ROS containers for a specified projection. `PARTITION_PROJECTION` also purges data while partitioning ROS containers if deletes were applied before the AHM epoch.

Syntax

```
PARTITION_PROJECTION ( '[schema.]projection-name'
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>projection-name</i>	The projection to partition.

Privileges

- Table owner
- USAGE privilege on schema

Examples

In this example, PARTITION_PROJECTION forces a split of ROS containers on the states_p projection:

```
=> SELECT PARTITION_PROJECTION ('states_p');
PARTITION_PROJECTION
-----
Projection partitioned
(1 row)
```

See Also

- [DO_TM_TASK](#)
- [DROP_PARTITION](#)
- [DUMP_PARTITION_KEYS](#)
- [DUMP_PROJECTION_PARTITION_KEYS](#)
- [DUMP_TABLE_PARTITION_KEYS](#)
- [PARTITION_TABLE](#)
- [Using Table Partitions](#)

PARTITION_TABLE

Forces the database to split ROS containers in the specified table that contain multiple distinct values of the partitioning expression. Only ROS containers with more than one distinct value can be split.

Syntax

```
PARTITION_TABLE ( '[schema.]table-name'
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	The table to partition.

Privileges

- Table owner
- USAGE privilege on schema

Restrictions

You cannot run `PARTITION_TABLE` on a table that is an anchor table for a live aggregate projection or a Top-K projection.

Examples

The following example creates a simple table `states` and partitions the data by state:

```
=> CREATE TABLE states (year INTEGER NOT NULL, state VARCHAR NOT NULL) PARTITION BY state;  
=> CREATE PROJECTION states_p (state, year) AS  
    SELECT * FROM states ORDER BY state, year UNSEGMENTED ALL NODES;
```

Run `PARTITION_TABLE` to partition the table `states`:

```
=> SELECT PARTITION_TABLE('states');
           PARTITION_TABLE
-----
Task: partition operation
(Table: public.states) (Projection: public.states_p)
(1 row)
```

See Also

- [DO_TM_TASK](#)
- [DROP_PARTITION](#)
- [DUMP_PARTITION_KEYS](#)
- [DUMP_PROJECTION_PARTITION_KEYS](#)
- [DUMP_TABLE_PARTITION_KEYS](#)
- [PARTITION_PROJECTION](#)
- [Using Table Partitions](#)

PURGE_PARTITION

Purges a table partition of deleted rows. Similar to `PURGE` and `PURGE_PROJECTION`, this function removes deleted data from physical storage so you can reuse the disk space. `PURGE_PARTITION` removes data only from the AHM epoch and earlier.

Syntax

```
PURGE_PARTITION ( '[schema.]table-name', partition-key )
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>table-name</i>	The partitioned table to purge.
<i>partition-key</i>	The key of the partition to purge.

Privileges

- Table owner
- USAGE privilege on schema

Example

The following example lists the count of deleted rows for each partition in a table, then calls `PURGE_PARTITION()` to purge the deleted rows from the data.

```
=> SELECT partition_key,table_schema,projection_name,sum(deleted_row_count)
AS deleted_row_count FROM partitions
GROUP BY partition_key,table_schema,projection_name
ORDER BY partition_key;
```

partition_key	table_schema	projection_name	deleted_row_count
0	public	t_super	2
1	public	t_super	2
2	public	t_super	2
3	public	t_super	2
4	public	t_super	2
5	public	t_super	2
6	public	t_super	2
7	public	t_super	2
8	public	t_super	2
9	public	t_super	1

(10 rows)

```
=> SELECT PURGE_PARTITION('t',5); -- Purge partition with key 5.
purge_partition
```

```
-----
Task: merge partitions
(Table: public.t) (Projection: public.t_super)
(1 row)
```

```
=> SELECT partition_key,table_schema,projection_name,sum(deleted_row_count)
AS deleted_row_count FROM partitions
GROUP BY partition_key,table_schema,projection_name
ORDER BY partition_key;
```

partition_key	table_schema	projection_name	deleted_row_count
0	public	t_super	2
1	public	t_super	2
2	public	t_super	2
3	public	t_super	2
4	public	t_super	2
5	public	t_super	0
6	public	t_super	2
7	public	t_super	2
8	public	t_super	2
9	public	t_super	1

(10 rows)

See Also

- [PURGE](#)
- [PURGE_PROJECTION](#)
- [PURGE_TABLE](#)
- [STORAGE_CONTAINERS](#)

SWAP_PARTITIONS_BETWEEN_TABLES

Swaps partitions between two tables.

Syntax

```
SWAP_PARTITIONS_BETWEEN_TABLES (  
    '[schema.]staging-table',  
    'min-range-value',  
    'max-range-value',  
    '[schema.]target-table'  
)
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>staging-table</i>	The staging table from which to swap partitions.
<i>min-range-value</i>	The minimum value of the partition to swap.
<i>max-range-value</i>	The maximum value of the partition to swap.
<i>target-table</i>	The table to which the partitions are to be swapped. The target table cannot be the same as the staging table.

Privileges

One of the following:

- [DBADMIN role](#)
- Owner of both tables
- USAGE privileges on both schemas, and TRUNCATE and INSERT privileges on both tables.

Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partitioning expression
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled. For more information on constraints, refer to [Enforcing Primary Key and Foreign Key Constraints](#).

Note: If the target table has primary or unique key constraints enabled and copying or moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation.

- Check constraints. For [MOVE_PARTITIONS_TO_TABLE](#) and [COPY_PARTITIONS_TO_TABLE](#), Vertica enforces enabled check constraints on the target table only. For [SWAP_PARTITIONS_BETWEEN_TABLES](#), Vertica enforces enabled check constraints on both tables. If there is a violation of an enabled check constraint, Vertica rolls back the operation.
- Number and definitions of text indices.

Table Restrictions

The following restrictions apply to the source and target tables:

- If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.
- The following tables cannot be used as sources or targets:
 - Temporary tables
 - Virtual tables
 - System tables
 - External tables

Examples

See [Swapping Partitions](#).

See Also

- [MOVE_PARTITIONS_TO_TABLE](#)
- [Swapping Partitions](#)
- [Tutorial for Swapping Partitions](#)

Profiling Functions

This section contains profiling functions specific to Vertica.

CLEAR_PROFILING

Clears from memory data for the specified profiling type.

Note: Vertica stores profiled data in memory, so profiling can be memory intensive depending on how much data you collect.

Syntax

```
CLEAR_PROFILING( 'profiling-type' )
```

Parameters

<i>profiling-type</i>	<p>The type of profiling data to clear:</p> <ul style="list-style-type: none">• session: Clears profiling for basic session parameters and lock time out data.• query: Clears profiling for general information about queries that ran, such as the query strings used and the duration of queries.• ee: Clears profiling for information about the execution run of each query.
-----------------------	---

Example

The following statement clears profiled data for queries:

```
=> SELECT CLEAR_PROFILING( 'query' );
```

See Also

- [DISABLE_PROFILING](#)
- [ENABLE_PROFILING](#)
- [SHOW_PROFILING_CONFIG](#)
- [Profiling Database Performance](#)

DISABLE_PROFILING

Disables profiling for the specified profiling type.

Syntax

```
DISABLE_PROFILING( 'profiling-type' )
```

Parameters

<i>profiling-type</i>	<p>The type of profiling data to disable:</p> <ul style="list-style-type: none">• session: Disables profiling for basic session parameters and lock time out data.• query: Disables profiling for general information about queries that ran, such as the query strings used and the duration of queries.• ee: Disables profiling for information about the execution run of each query.
-----------------------	---

Example

The following statement disables profiling on query execution runs:

```
=> SELECT DISABLE_PROFILING('ee');
      DISABLE_PROFILING
-----
EE Profiling Disabled
(1 row)
```

See Also

- [CLEAR_PROFILING](#)
- [ENABLE_PROFILING](#)
- [SHOW_PROFILING_CONFIG](#)
- [Profiling Database Performance](#)

ENABLE_PROFILING

Enables profiling for the specified profiling type.

Note: Vertica stores profiled data in memory, so profiling can be memory intensive depending on how much data you collect.

Syntax

```
ENABLE_PROFILING( 'profiling-type' )
```

Parameters

<i>profiling-type</i>	<p>The type of profiling data to enable:</p> <ul style="list-style-type: none">• session: Enables profiling for basic session parameters and lock time out data.• query: Enables profiling for general information about queries that ran, such as the query strings used and the duration of queries.• ee: Enables profiling for information about the execution run of each query.
-----------------------	---

Example

The following statement enables profiling on query execution runs:

```
=> SELECT ENABLE_PROFILING('ee');  
ENABLE_PROFILING
```

```
-----  
EE Profiling Enabled  
(1 row)
```

See Also

- [CLEAR_PROFILING](#)
- [DISABLE_PROFILING](#)
- [SHOW_PROFILING_CONFIG](#)
- [Profiling Database Performance](#)

SHOW_PROFILING_CONFIG

Shows whether profiling is enabled.

Syntax

```
SHOW_PROFILING_CONFIG ()
```

Example

The following statement shows that profiling is enabled globally for all profiling types (session, execution engine, and query):

```
=> SELECT SHOW_PROFILING_CONFIG();  
SHOW_PROFILING_CONFIG  
-----  
Session Profiling: Session off, Global on  
EE Profiling:      Session off, Global on  
Query Profiling:   Session off, Global on  
(1 row)
```

See Also

- [CLEAR_PROFILING](#)
- [DISABLE_PROFILING](#)

- [ENABLE_PROFILING](#)
- [Profiling Database Performance](#)

Projection Management Functions

This section contains projection management functions specific to Vertica.

See Also

- [V_CATALOG.PROJECTIONS](#)
- [V_CATALOG.PROJECTION_COLUMNS](#)
- [V_MONITOR.PROJECTION_REFRESHES](#)
- [V_MONITOR.PROJECTION_STORAGE](#)

CLEAR_PROJECTION_REFRESHES

Triggers Vertica to clear information about refresh operations for projections immediately.

Syntax

```
CLEAR_PROJECTION_REFRESHES()
```

Notes

Information about a refresh operation—whether successful or unsuccessful—is maintained in system table [PROJECTION_REFRESHES](#) until the function [CLEAR_PROJECTION_REFRESHES](#) executes or the storage quota for the table is exceeded. The table's `IS_EXECUTING` column returns a boolean value that indicates whether the refresh is running now (t) or occurred earlier (f).

Privileges

Superuser

Example

To immediately purge projection refresh history, use the `CLEAR_PROJECTION_REFRESHES()` function:

```
=> SELECT CLEAR_PROJECTION_REFRESHES();
CLEAR_PROJECTION_REFRESHES
-----
CLEAR
(1 row)
```

Only the rows where the `PROJECTION_REFRESHES.IS_EXECUTING` column equals false are cleared.

See Also

- [PROJECTION_REFRESHES](#)
- [REFRESH](#)
- [START_REFRESH](#)
- [Clearing PROJECTION_REFRESHES History](#)

EVALUATE_DELETE_PERFORMANCE

Evaluates projections for potential [DELETE](#) performance issues. If there are issues found, a warning message is displayed. For steps you can take to resolve delete and update performance issues, see [DELETE and UPDATE Optimization](#) in the Administrator's Guide. This function uses data sampling to determine whether there are any issues with a projection. Therefore, it does not generate false-positives warnings, but it can miss some cases where there are performance issues.

Note: Optimizing for delete performance is the same as optimizing for update performance. So, you can use this function to help optimize a projection for updates as well as deletes.

Syntax

```
EVALUATE_DELETE_PERFORMANCE ( 'target' )
```

Parameters

<i>target</i>	The name of a projection or table. If you supply the name of a projection, only that projection is evaluated for DELETE performance issues. If you supply the name of a table, then all of the projections anchored to the table will be evaluated for issues.
---------------	--

If you do not provide a projection or table name, `EVALUATE_DELETE_PERFORMANCE` examines all of the projections that you can access for DELETE performance issues. Depending on the size of your database, this may take a long time.

Privileges

None

Notes

When evaluating multiple projections, `EVALUATE_DELETE_PERFORMANCE` reports up to ten projections that have issues, and refers you to a table that contains the full list of issues it has found.

Example

The following example demonstrates how you can use `EVALUATE_DELETE_PERFORMANCE` to evaluate your projections for slow DELETE performance.

```
=> create table example (A int, B int,C int);
CREATE TABLE
=> create projection one_sort (A,B,C) as (select A,B,C from example) order by A;
CREATE PROJECTION
=> create projection two_sort (A,B,C) as (select A,B,C from example) order by A,B;
CREATE PROJECTION
=> select evaluate_delete_performance('one_sort');
           evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
=> select evaluate_delete_performance('two_sort');
           evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
```

The previous example showed that there was no structural issue with the projection that would cause poor DELETE performance. However, the data contained within the projection can create potential delete issues if the sorted columns do not uniquely identify a row or small number of rows. In the following example, Perl is used to populate the table with data using a nested series of loops. The inner loop populates column C, the middle loop populates column B, and the outer loop populates column A. The result is column A contains only three distinct values (0, 1, and 2), while column B slowly varies between 20 and 0 and column C changes in

each row. EVALUATE_DELETE_PERFORMANCE is run against the projections again to see if the data within the projections causes any potential DELETE performance issues.

```
=> \! perl -e 'for ($i=0; $i<3; $i++) { for ($j=0; $j<21; $j++) { for ($k=0; $k<19; $k++) { printf "%d,%d,%d\n", $i,$j,$k;}}}' | /opt/vertica/bin/vsql -c "copy example from stdin delimiter ',' direct;"
Password:
=> select * from example;
 A | B | C
-----+-----+-----
 0 | 20 | 18
 0 | 20 | 17
 0 | 20 | 16
 0 | 20 | 15
 0 | 20 | 14
 0 | 20 | 13
 0 | 20 | 12
 0 | 20 | 11
 0 | 20 | 10
 0 | 20 | 9
 0 | 20 | 8
 0 | 20 | 7
 0 | 20 | 6
 0 | 20 | 5
 0 | 20 | 4
 0 | 20 | 3
 0 | 20 | 2
 0 | 20 | 1
 0 | 20 | 0
 0 | 19 | 18
1157 rows omitted
 2 | 1 | 0
 2 | 0 | 18
 2 | 0 | 17
 2 | 0 | 16
 2 | 0 | 15
 2 | 0 | 14
 2 | 0 | 13
 2 | 0 | 12
 2 | 0 | 11
 2 | 0 | 10
 2 | 0 | 9
 2 | 0 | 8
 2 | 0 | 7
 2 | 0 | 6
 2 | 0 | 5
 2 | 0 | 4
 2 | 0 | 3
 2 | 0 | 2
 2 | 0 | 1
 2 | 0 | 0
=> SELECT COUNT (*) FROM example;
COUNT
-----
1197
(1 row)
=> SELECT COUNT (DISTINCT A) FROM example;
COUNT
-----
```

```
3
(1 row)
=> select evaluate_delete_performance('one_sort');
       evaluate_delete_performance
-----
Projection exhibits delete performance concerns.
(1 row)
release=> select evaluate_delete_performance('two_sort');
       evaluate_delete_performance
-----
No projection delete performance concerns found.
(1 row)
```

The `one_sort` projection has potential delete issues since it only sorts on column A which has few distinct values. This means that each value in the sort column corresponds to many rows in the projection, which negatively impacts DELETE performance. Since the `two_sort` projection is sorted on columns A and B, each combination of values in the two sort columns identifies just a few rows, allowing deletes to be performed faster.

Not supplying a projection name results in all of the projections you can access being evaluated for DELETE performance issues.

```
=> select evaluate_delete_performance();
       evaluate_delete_performance
-----
The following projection exhibits delete performance concerns:
"public"."one_sort"
See v_catalog.projection_delete_concerns for more details.
(1 row)
```

GET_PROJECTION_STATUS

Returns information relevant to the status of a projection:

- The current K-safety status of the database
- The number of nodes in the database
- Whether the projection is segmented
- The number and names of buddy projections
- Whether the projection is safe
- Whether the projection is up-to-date
- Whether statistics have been computed for the projection

Use `GET_PROJECTION_STATUS` to monitor the progress of a projection data refresh.

Syntax

```
GET_PROJECTION_STATUS ( '[schema-name.]projection' );
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>projection</i>	The projection for which to display status.

Examples

```
=> SELECT GET_PROJECTION_STATUS('public.customer_dimension_site01');
       GET_PROJECTION_STATUS
-----
Current system K is 1.
# of Nodes: 4.
public.customer_dimension_site01 [Segmented: No] [Seg Cols: ] [K: 3] [public.customer_dimension_
site04, public.customer_dimension_site03,
public.customer_dimension_site02]
[Safe: Yes] [UptoDate: Yes][Stats: Yes]
```

See Also

- [ALTER PROJECTION RENAME](#)
- [GET_PROJECTIONS](#)

GET_PROJECTIONS

Returns information about projections that are anchored to the specified table, as follows:

Contextual information	Projection data
<ul style="list-style-type: none"> • Database K-safety • The number of nodes in the database • Number of projections for this table 	<p>For each projection anchored to this table:</p> <ul style="list-style-type: none"> • The projection's buddy projections • Whether the projection is segmented

Contextual information	Projection data
	<ul style="list-style-type: none"> • Whether the projection is safe • Whether the projection is up-to-date

You can use GET_PROJECTIONS to monitor the progress of a projection data refresh.

Syntax

```
GET_PROJECTIONS ( '[schema-name.]table' )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table</i>	The table whose projections you want to list.

Privileges

None

Examples

The following example gets information about the `store_dimension` table in the `VMart` schema:

```
=> SELECT GET_PROJECTIONS('store.store_dimension');
           GET_PROJECTIONS
-----
Current system K is 1.
# of Nodes: 3.
Table store.store_dimension has 3 projections.

Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate] [Stats]
-----
store.store_dimension_unseg [Segmented: No] [Seg Cols: ] [K: 2] [store.store_dimension_unseg] [Safe:
Yes] [UptoDate: Yes] [Stats: Yes]
store.store_dimension_p_b1 [Segmented: Yes] [Seg Cols: "store.store_dimension.store_key"] [K: 1]
[store.store_dimension_p_b0] [Safe: Yes] [UptoDate: Yes] [Stats: Yes]
store.store_dimension_p_b0 [Segmented: Yes] [Seg Cols: "store.store_dimension.store_key"] [K: 1]
```

```
[store.store_dimension_p_b1] [Safe: Yes] [UptoDate: Yes] [Stats: Yes]
```

```
(1 row)
```

REFRESH

Performs a synchronous, optionally-targeted refresh of a specified table's projections.

Information about a refresh operation—whether successful or unsuccessful—is maintained in system table [PROJECTION_REFRESHES](#) until the function [CLEAR_PROJECTION_REFRESHES](#) executes or the storage quota for the table is exceeded. The table's `IS_EXECUTING` column returns a boolean value that indicates whether the refresh is running now (t) or occurred earlier (f).

Syntax

```
REFRESH ( '[schema.]table-name[,...]' )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	<p>The anchor table of the projections to refresh. If you specify multiple tables, REFRESH attempts to refresh them in parallel. Such calls will be part of the Database Designer deployment (and deployment script).</p>

Returns

Column Name	Description
Projection Name	The projection targeted for refresh.
Anchor Table	The projection's associated anchor table.
Status	<p>Status of the projection:</p> <ul style="list-style-type: none"> Queued: A projection is queued for refresh.

	<ul style="list-style-type: none"> • Refreshing: A projection refresh is in process. • Refreshed: A projection refresh successfully completed. • Failed: A projection refresh did not successfully complete.
Refresh Method	<p>Method used to refresh the projection:</p> <ul style="list-style-type: none"> • Buddy: Uses the contents of a buddy to refresh the projection. This method maintains historical data. This enables the projection to be used for historical queries. • Scratch: Refreshes the projection without using a buddy. This method does not generate historical data. This means that the projection cannot participate in historical queries from any point before the projection was refreshed.
Error Count	Number of times a refresh failed for the projection.
Duration (sec)	Length of time that the projection refresh ran in seconds.

Privileges

- Superuser
- Owner of the specified tables

Notes

- Unlike [START_REFRESH](#) which runs in the background, `REFRESH()` runs in the foreground of the caller's session.
- If you run `REFRESH()` without arguments, it refreshes all non up-to-date projections. If the function returns a header string with no results, then no projections needed refreshing.

Examples

The following example refreshes the projections in tables `t1` and `t2`:

```
=> SELECT REFRESH('t1, t2');
                                REFRESH
-----
Refresh completed with the following outcomes:
```

```
Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]
-----
"public"."t1_p": [t1] [refreshed] [scratch] [0] [0]"public"."t2_p": [t2] [refreshed] [scratch] [0]
[0]
```

This next example shows that only the projection on table `t` was refreshed:

```
=> SELECT REFRESH('allow, public.deny, t');
                                REFRESH
-----
Refresh completed with the following outcomes:

Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]
-----
"n/a"."n/a": [n/a] [failed: insufficient permissions on table "allow"] [] [1] [0]
"n/a"."n/a": [n/a] [failed: insufficient permissions on table "public.deny"] [] [1] [0]
"public"."t_p1": [t] [refreshed] [scratch] [0] [0]
```

See Also

- [CLEAR_PROJECTION_REFRESHES](#)
- [PROJECTION_REFRESHES](#)
- [START_REFRESH](#)
- [Clearing PROJECTION_REFRESHES History](#)

REFRESH_COLUMNS

Refreshes table columns that are defined with the constraint `SET USING`. All refresh operations associated with a `REFRESH_COLUMNS` operation belong to the same transaction. All tables and columns specified by `REFRESH_COLUMNS` must be refreshed; otherwise, the entire operation is rolled back.

Syntax

```
REFRESH_COLUMNS ( 'tables', '[columns]' [, '[refresh-mode]' ] )
```

Parameters

<i>tables</i>	A comma-delimited list that specifies the tables to refresh:
---------------	--

	<p><code>[<i>schema.</i>]table[,...]</code></p> <p>Important: If you specify multiple tables, you must also set parameter <i>refresh-mode</i> to REBUILD.</p>
<i>columns</i>	<p>Specifies one or more columns to refresh, as follows:</p> <ul style="list-style-type: none"> • ' ' (empty string) Refresh all SET USING/DEFAULT USING columns in the specified tables. • <code>[[<i>schema.</i>]table.]column[,...]</code> Refresh all columns in the comma-delimited list. The following requirements apply: <ul style="list-style-type: none"> ■ If REFRESH_COLUMNS specifies multiple tables, all column names must be qualified by their table names. If the target tables span multiple schemas, all column names must be fully qualified by their schema and table names. ■ All specified columns must have a SET USING or DEFAULT USING constraint. <p>For example:</p> <pre>SELECT REFRESH_COLUMNS ('t1, t2', 't1.a, t2.b', 'REBUILD');</pre> • <code>[<i>schema.</i>]table.*</code> Refresh all SET USING/DEFAULT USING columns in <i>table</i>. For example: <pre>SELECT REFRESH_COLUMNS ('t1, t2', 't1.*, t2.b', 'REBUILD');</pre>
<i>refresh-mode</i>	<p>Specifies how to refresh SET USING columns:</p> <ul style="list-style-type: none"> • UPDATE: Marks original rows as deleted and replaces them with new rows. In order to save these updates, you must issue a COMMIT statement. • REBUILD: Replaces all data in the specified columns. The rebuild operation is auto-committed. <p>If set to an empty string or omitted, REFRESH_COLUMNS executes in UPDATE mode. If you specify multiple tables, you must explicitly specify</p>

	<p>REBUILD mode.</p> <p>In both cases, <code>REFRESH_COLUMNS</code> returns an error if any <code>SET_USING</code> column is defined as a primary or unique key in a table that enforces those constraints.</p> <p>See REBUILD Mode Restrictions for limitations on using the <code>REBUILD</code> option.</p>
--	--

Privileges

- `MODIFY` privilege on the target table, `USAGE` privilege on its schema
- For each `SET USING` column to refresh that queries another table or view: `SELECT` privilege on the queried table/view, `USAGE` privilege on its schema

REBUILD versus REFRESH Modes

In general, `UPDATE` is a better choice when changes to `SET USING` column data are confined to a relatively small number of rows. Use `REBUILD` when a significant amount of `SET USING` column data is stale and must be updated. It is generally good practice to call `REFRESH_COLUMNS` with `REBUILD` on any new `SET USING` column—for example, to populate a `SET USING` column after adding it with `ALTER TABLE . . . ADD COLUMN`.

REBUILD Mode Restrictions

If you call `REFRESH_COLUMNS` on a `SET USING` column and specify the refresh mode as `REBUILD`, Vertica returns an error if any of the following conditions is true for that column:

- Specified as a table partition key.
- Included in a [live aggregate projection](#) or [projection with expressions](#).
- Included in a projection's sort order or segmentation.
- Included in a projection, and the projection omits an anchor table column that is referenced in the column's `SET USING` expression.
- Included in a projection's `GROUPED clause`.

See Also

- [Column-Constraint](#)
- [Defining Column Values](#)

START_REFRESH

For the current schema, transfers data to projections that are not able to participate in query execution due to missing or out-of-date data.

Syntax

```
START_REFRESH()
```

Notes

- If you want to refresh only the projections in a specific table, use [REFRESH](#).
- Unlike [REFRESH\(\)](#), which runs in the foreground of the caller's session, [START_REFRESH\(\)](#) runs in the background.
- When a design is deployed through the Database Designer, it is automatically refreshed. See [Deploying a Design](#) in the Administrator's Guide.
- All nodes must be up in order to start a refresh.
- [START_REFRESH\(\)](#) has no effect if a refresh is already running.
- The refresh runs asynchronously.
- Shutting down the database ends the refresh.
- To view the progress of the refresh, see the [PROJECTION_REFRESHES](#) and [PROJECTIONS](#) system tables.
- If a projection is updated from scratch, the data stored in the projection represents the table columns as of the epoch in which the refresh commits. As a result, the query optimizer might not choose the new projection for AT EPOCH queries that request historical data at epochs older than the refresh epoch of the projection. Projections refreshed from buddies retain history and can be used to answer historical queries.

Privileges

None

Example

The following command starts the refresh operation:

```
=> SELECT START_REFRESH();
      start_refresh
-----
Starting refresh background process.
```

See Also

- [CLEAR_PROJECTION_REFRESHES](#)
- [MARK_DESIGN_KSAFE](#)
- [PROJECTION_REFRESHES](#)
- [PROJECTIONS](#)
- [Clearing PROJECTION_REFRESHES History](#)

Purge Functions

This section contains purge functions specific to Vertica.

PURGE

Permanently removes delete vectors from ROS storage containers so disk space can be reused. PURGE removes all historical data up to and including the Ancient History Mark epoch.

PURGE does not delete temporary tables.

Caution: PURGE can temporarily take up significant disk space.

Syntax

```
PURGE()
```

Privileges

- Table owner
- USAGE privilege on schema

See Also

- [PURGE_PROJECTION](#)
- [PURGE_TABLE](#)
- [Purging Deleted Data](#)

PURGE_PROJECTION

Permanently removes deleted data from physical storage so disk space can be reused. You can purge historical data up to and including the Ancient History Mark epoch.

Caution: PURGE_PROJECTION can use significant disk space while purging the data.

See [PURGE](#) for details about purge operations.

Syntax

```
PURGE_PROJECTION ( '[schema.]projection-name' )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>projection-name</i>	The projection to purge.

Privileges

- Table owner
- USAGE privilege on schema

Examples

The following example purges all historical data in projection `tbl_p` that precedes the Ancient History Mark epoch.

```
=> CREATE TABLE tbl (x int, y int);
CREATE TABLE
=> INSERT INTO tbl VALUES(1,2);
  OUTPUT
  -----
         1
(1 row)

=> INSERT INTO tbl VALUES(3,4);
  OUTPUT
  -----
         1
(1 row)

dbadmin=> COMMIT;
COMMIT
=> CREATE PROJECTION tbl_p AS SELECT x FROM tbl UNSEGMENTED ALL NODES;
WARNING 4468: Projection <public.tbl_p> is not available for query processing.
Execute the select start_refresh() function to copy data into this projection.
The projection must have a sufficient number of buddy projections and all nodes must be up before
starting a refresh
CREATE PROJECTION
```

```
=> SELECT START_REFRESH();
      START_REFRESH
-----
Starting refresh background process.
=> DELETE FROM tbl WHERE x=1;
OUTPUT
-----
      1
(1 row)

=> COMMIT;
COMMIT
=> SELECT MAKE_AHM_NOW();
      MAKE_AHM_NOW
-----
AHM set (New AHM Epoch: 9066)
(1 row)

=> SELECT PURGE_PROJECTION ('tbl_p');
      PURGE_PROJECTION
-----
Projection purged
(1 row)
```

See Also

- [PURGE_TABLE](#)
- [STORAGE_CONTAINERS](#)
- [Purging Deleted Data](#) in the Administrator's Guide.

PURGE_TABLE

Note: This function was formerly named `PURGE_TABLE_PROJECTIONS()`. Vertica still supports the former function name.

Permanently removes deleted data from physical storage so disk space can be reused. You can purge historical data up to and including the Ancient History Mark epoch.

Purges all projections of the specified table. You cannot use this function to purge temporary tables.

Syntax

```
PURGE_TABLE ( '[schema.]table-name' )
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>table-name</i>	The table to purge.

Privileges

- Table owner
- USAGE privilege on schema

Caution: PURGE_TABLE could temporarily take up significant disk space while the data is being purged.

Example

The following example purges all projections for the store sales fact table located in the Vmart schema:

```
=> SELECT PURGE_TABLE('store.store_sales_fact');
```

See Also

- [PURGE](#)
- [PURGE_TABLE](#)
- [STORAGE_CONTAINERS](#)
- [Purging Deleted Data](#)

Session Management Functions

This section contains session management functions specific to Vertica.

See also the SQL system table [V_MONITOR.SESSIONS](#).

CANCEL_REFRESH

Cancels refresh-related internal operations initiated by [START_REFRESH](#) and [REFRESH](#).

Syntax

```
CANCEL_REFRESH()
```

Privileges

None

Notes

- Refresh tasks run in a background thread in an internal session, so you cannot use [INTERRUPT_STATEMENT](#) to cancel those statements. Instead, use `CANCEL_REFRESH` to cancel statements that are run by refresh-related internal sessions.
- Run `CANCEL_REFRESH()` on the same node on which `START_REFRESH()` was initiated.
- `CANCEL_REFRESH()` cancels the refresh operation running on a node, waits for the cancelation to complete, and returns `SUCCESS`.
- Only one set of refresh operations runs on a node at any time.

Example

Cancel a refresh operation executing in the background.

```
=> SELECT START_REFRESH();
      START_REFRESH
-----
Starting refresh background process.
```

```
(1 row)
=> SELECT CANCEL_REFRESH();
       CANCEL_REFRESH
-----
Stopping background refresh process.
(1 row)
```

See Also

- [INTERRUPT_STATEMENT](#)
- [SESSIONS](#)
- [START_REFRESH](#)
- [PROJECTION_REFRESHES](#)

CLOSE_ALL_SESSIONS

Closes all external sessions except the one that issues this function. Use `CLOSE_ALL_SESSIONS` before [shutting down](#) the Vertica database.

Vertica closes sessions asynchronously, so it is possible for another session to open before all previous sessions close. In this case, you must reissue this function before the database can be shut down. To view the status of all open sessions, query the system table [SESSIONS](#).

Query to view all database sessions.

For detailed information about session management options, see [Managing Sessions](#) in the Administrator's Guide.

Syntax

```
CLOSE_ALL_SESSIONS()
```

Privileges

- Superuser to close all database sessions
- None to close your own session

Examples

Two user sessions are open on separate nodes:

```
=> SELECT * FROM sessions;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node0002
user_name          | dbadmin
client_hostname    | 127.0.0.1:57174
client_pid         | 30117
login_timestamp    | 2011-01-03 15:33:00.842021-05
session_id         | stress05-27944:0xc1a
client_label       |
transaction_start  | 2011-01-03 15:34:46.538102
transaction_id     | -1
transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/mart_Fact.tbl'
DELIMITER '|' NULL '\\n');
statement_start    | 2011-01-03 15:34:46.538862
statement_id       |
last_statement_duration_us | 26250
current_statement  | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER '|'
NULL '\\n';
ssl_state          | None
authentication_method | Trust
-[ RECORD 3 ]-----+-----
node_name          | v_vmartdb_node0003
user_name          | dbadmin
client_hostname    | 127.0.0.1:56367
client_pid         | 1191
login_timestamp    | 2011-01-03 15:31:44.939302-05
session_id         | stress06-25663:0xbec
client_label       |
transaction_start  | 2011-01-03 15:34:51.05939
transaction_id     | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/Mart_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;)
statement_start    | 2011-01-03 15:35:46.436748
statement_id       |
last_statement_duration_us | 1591403
current_statement  | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER '|'
NULL '\\n' DIRECT;
ssl_state          | None
authentication_method | Trust
```

Close all sessions:

```
=> \x
Expanded display is off.
=> SELECT CLOSE_ALL_SESSIONS();
                CLOSE_ALL_SESSIONS
-----
Close all sessions command sent. Check v_monitor.sessions for progress.
(1 row)
```

Session contents after issuing CLOSE_ALL_SESSIONS:

```
=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start    | 2011-01-03 16:19:56.720071
statement_id       | 25
last_statement_duration_us | 15605
current_statement  | SELECT * FROM SESSIONS;
ssl_state          | None
authentication_method | Trust
```

See Also

- [CLOSE_SESSION](#)
- [CLOSE_USER_SESSIONS](#)
- [SHUTDOWN](#)

CLOSE_SESSION

Interrupts the specified external session, rolls back the current transaction if any, and closes the socket. You can only close your own session.

It might take some time before a session is closed. To view the status of all open sessions, query the system table [SESSIONS](#).

For detailed information about session management options, see [Managing Sessions](#) in the Administrator's Guide.

Syntax

```
CLOSE_SESSION ( 'sessionid')
```

Parameters

<i>sessionid</i>	A string that specifies the session to close. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
------------------	--

Privileges

None

Examples

User session opened. Record 2 shows the user session running a COPY DIRECT statement.

```
=> SELECT * FROM sessions;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node0002
user_name          | dbadmin
client_hostname    | 127.0.0.1:57174
client_pid         | 30117
login_timestamp    | 2011-01-03 15:33:00.842021-05
session_id         | stress05-27944:0xc1a
client_label       |
transaction_start  | 2011-01-03 15:34:46.538102
transaction_id     | -1
transaction_description | user dbadmin (COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;)
statement_start    | 2011-01-03 15:34:46.538862
```

```
statement_id          |
last_statement_duration_us | 26250
current_statement     | COPY ClickStream_Fact FROM '/data/clickstream
                        | /1g/ClickStream_Fact.tbl' DELIMITER '|' NULL
                        | '\\n' DIRECT;
ssl_state             | None
authentication_method | Trust
```

Close user session stress05-27944:0xc1a

```
=> \x
Expanded display is off.
=> SELECT CLOSE_SESSION('stress05-27944:0xc1a');
                CLOSE_SESSION
-----
Session close command sent. Check v_monitor.sessions for progress.
(1 row)
```

Query the sessions table again for current status, and you can see that the second session has been closed:

```
=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from SESSIONS;)
statement_start    | 2011-01-03 16:12:07.841298
statement_id       | 20
last_statement_duration_us | 2099
current_statement  | SELECT * FROM SESSIONS;
ssl_state          | None
authentication_method | Trust
```

See Also

- [CLOSE_ALL_SESSIONS](#)
- [SHUTDOWN](#)

CLOSE_USER_SESSIONS

Stops the session for a user, rolls back any transaction currently running, and closes the connection. To determine the status of the sessions to close, query the [SESSIONS](#) table.

Note: Running this function on your own sessions leaves one session running.

Syntax

```
CLOSE_USER_SESSIONS ( 'user-name' )
```

Parameters

<i>user-name</i>	Specifies the user whose sessions are to be closed. If you specify your own user name, Vertica closes all sessions except the one in which you issue this function.
------------------	---

Privileges

[DBADMIN](#)

Examples

This example closes all active session for user u1:

```
=> SELECT close_user_sessions('u1');
```

See Also

- [CLOSE_ALL_SESSIONS](#)
- [CLOSE_SESSION](#)
- [SHUTDOWN](#)

GET_NUM_ACCEPTED_ROWS

Returns the number of rows loaded into the database for the last completed load for the current session. `GET_NUM_ACCEPTED_ROWS` is a meta-function. Do not use it as a value in an `INSERT` query.

The number of accepted rows is not available for a load that is currently in process. Check the [LOAD_STREAMS](#) system table for its status.

This meta-function supports loads from STDIN, COPY LOCAL from a Vertica client, or a single file on the initiator. You cannot use GET_NUM_ACCEPTED_ROWS for multi-node loads.

Syntax

```
GET_NUM_ACCEPTED_ROWS();
```

Privileges

None

Note: The data regarding accepted rows from the last load during the current session does not persist, and is lost when you initiate a new load.

Examples

This examples shows the number of accepted rows from the vmart_load_data.sql meta-command.

```
=> \i vmart_load_data.sql;
=> SELECT GET_NUM_ACCEPTED_ROWS ();
GET_NUM_ACCEPTED_ROWS
-----
300000
(1 row)
```

See Also

- [GET_NUM_REJECTED_ROWS](#)

GET_NUM_REJECTED_ROWS

Returns the number of rows that were rejected during the last completed load for the current session. GET_NUM_REJECTED_ROWS is a meta-function. Do not use it as a value in an INSERT query.

Rejected row information is unavailable for a load that is currently running. The number of rejected rows is not available for a load that is currently in process. Check the [LOAD_STREAMS](#) system table for its status.

This meta-function supports loads from STDIN, COPY LOCAL from a Vertica client, or a single file on the initiator. You cannot use GET_NUM_REJECTED_ROWS for multi-node loads.

Syntax

```
GET_NUM_REJECTED_ROWS();
```

Privileges

None

Note: The data regarding rejected rows from the last load during the current session does not persist, and is dropped when you initiate a new load.

Examples

This example shows the number of rejected rows from the `vmart_load_data.sql` meta-command.

```
=> \i vmart_load_data.sql
=> SELECT GET_NUM_REJECTED_ROWS ();
GET_NUM_REJECTED_ROWS
-----
0
(1 row)
```

See Also

- [GET_NUM_ACCEPTED_ROWS](#)

INTERRUPT_STATEMENT

Interrupts the specified statement in a user session, rolls back the current transaction, and writes a success or failure message to the log file.

Sessions can be interrupted during statement execution. Only statements run by user sessions can be interrupted.

Syntax

```
INTERRUPT_STATEMENT( 'session-id', statement-id )
```

Parameters

<i>session-id</i>	Identifies the session to interrupt. This identifier is unique within the cluster at any point in time.
<i>statement-id</i>	Identifies the statement to interrupt. If the <i>statement-id</i> is valid, the statement can be interrupted and INTERRUPT_STATEMENT returns a success message. Otherwise the system returns an error.

Privileges

Superuser

Messages

The following list describes messages you might encounter:

Message	Meaning
Statement interrupt sent. Check SESSIONS for progress.	This message indicates success.
Session <id> could not be successfully interrupted: session not found.	The session ID argument to the interrupt command does not match a running session.
Session <id> could not be successfully interrupted: statement not found.	The statement ID does not match (or no longer matches) the ID of a running statement (if any).
No interruptible statement running	The statement is DDL or otherwise non-interruptible.
Internal (system) sessions cannot be interrupted.	The session is internal, and only statements run by external

Message	Meaning
	sessions can be interrupted.

Examples

Two user sessions are open. RECORD 1 shows user session running SELECT FROM SESSION, and RECORD 2 shows user session running COPY DIRECT:

```
=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
user_name          | dbadmin
client_hostname    | 127.0.0.1:52110
client_pid         | 4554
login_timestamp    | 2011-01-03 14:05:40.252625-05
session_id         | stress04-4325:0x14
client_label       |
transaction_start  | 2011-01-03 14:05:44.325781
transaction_id     | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start    | 2011-01-03 15:36:13.896288
statement_id       | 10
last_statement_duration_us | 14978
current_statement  | select * from sessions;
ssl_state          | None
authentication_method | Trust
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node0003
user_name          | dbadmin
client_hostname    | 127.0.0.1:56367
client_pid         | 1191
login_timestamp    | 2011-01-03 15:31:44.939302-05
session_id         | stress06-25663:0xbec
client_label       |
transaction_start  | 2011-01-03 15:34:51.05939
transaction_id     | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/Mart_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;)
statement_start    | 2011-01-03 15:35:46.436748
statement_id       | 5
last_statement_duration_us | 1591403
current_statement  | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER '|'
NULL '\\n' DIRECT;
ssl_state          | None
authentication_method | Trust
```

Interrupt the COPY DIRECT statement running in session stress06-25663:0xbec:

```
=> \x
Expanded display is off.
=> SELECT INTERRUPT_STATEMENT('stress06-25663:0x1537', 5);
      interrupt_statement
```

```
-----  
Statement interrupt sent. Check v_monitor.sessions for progress.  
(1 row)
```

Verify that the interrupted statement is no longer active by looking at the `current_statement` column in the `SESSIONS` system table. This column becomes blank when the statement is interrupted:

```
=> SELECT * FROM SESSIONS;  
-[ RECORD 1 ]-----+-----  
node_name          | v_vmartdb_node001  
user_name          | dbadmin  
client_hostname    | 127.0.0.1:52110  
client_pid         | 4554  
login_timestamp    | 2011-01-03 14:05:40.252625-05  
session_id         | stress04-4325:0x14  
client_label       |  
transaction_start  | 2011-01-03 14:05:44.325781  
transaction_id     | 45035996273728326  
transaction_description | user dbadmin (select * from sessions;)  
statement_start    | 2011-01-03 15:36:13.896288  
statement_id       | 10  
last_statement_duration_us | 14978  
current_statement  | select * from sessions;  
ssl_state          | None  
authentication_method | Trust  
-[ RECORD 2 ]-----+-----  
node_name          | v_vmartdb_node003  
user_name          | dbadmin  
client_hostname    | 127.0.0.1:56367  
client_pid         | 1191  
login_timestamp    | 2011-01-03 15:31:44.939302-05  
session_id         | stress06-25663:0xbec  
client_label       |  
transaction_start  | 2011-01-03 15:34:51.05939  
transaction_id     | 54043195528458775  
transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/Mart_Fact.tbl'  
DELIMITER '|' NULL '\\n' DIRECT;)  
statement_start    | 2011-01-03 15:35:46.436748  
statement_id       | 5  
last_statement_duration_us | 1591403  
current_statement  |  
ssl_state          | None  
authentication_method | Trust
```

See Also

- [SESSIONS](#)
- [Managing Sessions](#)
- [Configuration Parameters](#)

RELEASE_ALL_JVM_MEMORY

Forces all sessions to release the memory consumed by their Java Virtual Machines (JVM).

Syntax

```
RELEASE_ALL_JVM_MEMORY();
```

Privileges

Must be a superuser.

Example

The following example demonstrates viewing the JVM memory use in all open sessions, then calling `RELEASE_ALL_JVM_MEMORY()` to release the memory:

```
=> select user_name,external_memory_kb FROM V_MONITOR.SESSIONS;
  user_name | external_memory_kb
-----+-----
  dbadmin   |          79705
(1 row)

=> SELECT RELEASE_ALL_JVM_MEMORY();
                RELEASE_ALL_JVM_MEMORY
-----+-----
  Close all JVM sessions command sent. Check v_monitor.sessions for progress.
(1 row)

=> SELECT user_name,external_memory_kb FROM V_MONITOR.SESSIONS;
  user_name | external_memory_kb
-----+-----
  dbadmin   |          0
(1 row)
```

See Also

- [RELEASE_JVM_MEMORY](#)

RELEASE_JVM_MEMORY

Terminates a Java Virtual Machine (JVM), making available the memory the JVM was using.

Syntax

```
RELEASE_JVM_MEMORY();
```

Privileges

None.

Examples

User session opened. RECORD 2 shows the user session running COPY DIRECT statement.

```
=> SELECT RELEASE_JVM_MEMORY();
      release_jvm_memory
-----
Java process killed and memory released
(1 row)
```

See Also

- [RELEASE_ALL_JVM_MEMORY](#)

RESERVE_SESSION_RESOURCE

Reserves memory resources from the general resource pool for the exclusive use of the Vertica backup and restore process. No other Vertica process can access reserved resources. If insufficient resources are available, Vertica queues the reservation request.

This metafunction is a session level reservation. When a session ends Vertica automatically releases any resources reserved in that session. Because the metafunction operates at the session level, the resource name does not need to be unique across multiple sessions.

You can view reserved resources by querying the [SESSIONS](#) table.

Syntax

```
RESERVE_SESSION_RESOURCE ( 'name',memory_kb)
```

Parameters

<i>name</i>	[Required] Specifies the name of the resource that you want to reserve.
<i>memory_kb</i>	[Required] Specifies the amount of memory to allocate to the resource.

Privileges

None

Example

This example reserves 1024 kilobytes of memory for the backup and restore process:

```
=> SELECT reserve_session_resource('VBR_RESERVE',1024);
-[ RECORD 1 ]-----+-----
reserve_session_resource | Grant succeed
```

See Also

[RELEASE_SESSION_RESOURCE](#)

RESET_SESSION

Applies your default connection string configuration settings to your current session.

Syntax

```
RESET_SESSION()
```

Examples

The following example shows how you use `RESET_SESSION`.

Resets the current client connection string to the default connection string settings:

```
=> SELECT RESET_SESSION();
RESET_SESSION
-----
Reset session: done.
(1 row)
```

Statistic Management Functions

This section contains statistic management functions specific to Vertica.

ANALYZE_EXTERNAL_ROW_COUNT

Calculates the exact number of rows in an external table. ANALYZE_EXTERNAL_ROW_COUNT runs in the background.

Note: You cannot calculate row counts on external tables with [DO_TM_TASK](#).

Syntax

```
ANALYZE_EXTERNAL_ROW_COUNT ('[ [schema.]table-name ]')
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	<p>Specifies the name of the external table for which to calculate the exact row count. If you supply an empty string, Vertica calculate the exact number of rows for all external tables.</p>

Privileges

Any INSERT/UPDATE/DELETE privilege on the external table

Examples

Calculate the exact row count for all external tables:

```
=> SELECT ANALYZE_EXTERNAL_ROW_COUNT('');
```

Calculate the exact row count for table loader_rejects:

```
=> SELECT ANALYZE_EXTERNAL_ROW_COUNT('loader_rejects');
```

See Also

- [Collecting Database Statistics](#)
- [DROP_EXTERNAL_ROW_COUNT](#)

ANALYZE_STATISTICS

Note: ANALYZE_STATISTICS is an alias of the function ANALYZE_HISTOGRAM, which is no longer documented.

Collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table. By default, Vertica analyzes multiple columns in a single-query execution plan, depending on resource limits. Such multi-column analysis facilitates the following objectives:

- Reduce plan execution latency.
- Speed up analysis of relatively small tables with many columns.

Vertica writes statistics to the database catalog. The query optimizer uses this collected data to create query plans. Without this data, the query optimizer assumes uniform distribution of data values and equal storage usage for all projections.

You can cancel statistics collection with CTRL+C or by calling [INTERRUPT_STATEMENT](#).

Syntax

```
ANALYZE_STATISTICS ( '[ scope ]' [, 'column[,...]' [, percent ] )
```

Returns

0—Success

If an error occurs, refer to `vertica.log` for details.

Parameters

<i>scope</i>	<p>Specifies the table on which to collect data, as follows:</p> <p><i>[schema.]table</i></p> <p>If set to an empty string, Vertica collects statistics for all database tables and their projections.</p>
<i>column</i>	<p>The name of a column in <i>table</i>, typically a predicate column. You can specify multiple comma-delimited columns. Vertica narrows the scope of the data collection to the specified columns.</p> <p>If you alter a table to add a column and populate its contents with either default or other values, call <code>ANALYZE_STATISTICS</code> on this column to get the most current statistics.</p>
<i>percent</i>	<p>A float value between 0 and 100 that specifies what percentage of data to read from disk (not the amount of data to analyze). If you omit this argument, Vertica sets the percentage to 10.</p> <p>Analyzing more than 10 percent disk space takes proportionally longer to process, but produces a higher level of sampling accuracy.</p>

Privileges

- Any INSERT/UPDATE/DELETE privilege on the specified table.
- USAGE privilege on schema that contains the table

Restrictions

- Vertica collects no statistics on live aggregate and Top-K projections that are anchored to the specified table.
- If you include a SQL function within an expression when you create a projection, Vertica collects no statistics for that projection.
- Vertica supports `ANALYZE_STATISTICS` on [local temporary tables](#) but not on global temporary tables. To obtain statistics on a temporary table, create the table with the option `ON COMMIT PRESERVE ROWS`. Otherwise, Vertica deletes the table content when it commits the current transaction, so no table data is available for analysis.

See Also

- [Getting Statistics](#)
- [DROP_STATISTICS](#)
- [EXPORT_STATISTICS](#)
- [IMPORT_STATISTICS](#)
- [VALIDATE_STATISTICS](#)

DROP_EXTERNAL_ROW_COUNT

Removes external table row count statistics compiled by [ANALYZE_EXTERNAL_ROW_COUNT](#). DROP_EXTERNAL_ROW_COUNT runs in the background.

Caution: Statistics can be time consuming to regenerate.

Syntax

```
DROP_EXTERNAL_ROW_COUNT ('[ schema.]table-name ');
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	<p>The external table for which to remove the exact row count. If you specify an empty string, Vertica drops the exact row count statistic for all external tables.</p>

Privileges

- INSERT/UPDATE/DELETE privilege on table
- USAGE privilege on schema that contains the table

Examples

Drop row count statistics for external table `loader_rejects`:

```
=> SELECT DROP_EXTERNAL_ROW_COUNT('loader_rejects');
```

See Also

[Collecting Database Statistics](#)

DROP_STATISTICS

Removes statistical data on database projections previously generated by [ANALYZE_STATISTICS](#). When you drop this data, the Vertica optimizer uses default statistics to create query plans.

Caution: Before you drop statistics, be aware that sstatistics can be time consuming to regenerate.

Syntax

```
DROP_STATISTICS ('[ scope ]'[, 'category' ][, 'column[,...]' )
```

Parameters

<i>scope</i>	Specifies the target table as follows: <i>[schema.]table</i> If set to an empty string, Vertica drops statistics for projections of all database tables.
<i>category</i>	The category of statistics to drop for the named table, one of the following: <ul style="list-style-type: none">• BASE (default) drops histograms and row counts (min/max column values, histogram).• HISTOGRAMS drops only histograms. Row counts statistics remain.• ALL drops all statistics.

<i>column</i>	The name of a column in <i>table</i> , typically a predicate column. You can specify multiple comma-delimited columns. Vertica narrows the scope of dropped statistics to the specified columns only.
---------------	---

Privileges

- INSERT/UPDATE/DELETE privilege on table
- USAGE privilege on schema that contains the table

Examples

Drop all base statistics for the table `store.store_sales_fact`:

```
=> SELECT DROP_STATISTICS('store.store_sales_fact');
DROP_STATISTICS
-----
                0
(1 row)
```

Drop statistics for all table projections:

```
=> SELECT DROP_STATISTICS ('');
DROP_STATISTICS
-----
                0
(1 row)
```

See Also

- [ANALYZE_STATISTICS](#)
- [EXPORT_STATISTICS](#)
- [IMPORT_STATISTICS](#)
- [VALIDATE_STATISTICS](#)

EXPORT_STATISTICS

Generates database statistics in XML format from data previously collected by [ANALYZE_STATISTICS](#). Before you export statistics, collect the latest data by calling [ANALYZE_STATISTICS](#).

Syntax

```
EXPORT_STATISTICS ('[ filename ]'[, scope ]' [, 'column[,...]'
```

Parameters

<i>filename</i>	Specifies where to write the generated XML. If <i>filename</i> already exists, EXPORT_STATISTICS overwrites it. If you supply an empty string, EXPORT_STATISTICS writes the XML to standard output.
<i>scope</i>	Specifies the target table as follows: <i>[schema.]table</i> If you specify a table, Vertica exports statistics for its projections. If you supply an empty string, Vertica exports all statistics for the database.
<i>column</i>	The name of a column in <i>table</i> , typically a predicate column. You can specify multiple comma-delimited columns. Vertica narrows the scope of exported statistics to the specified columns only.

Privileges

Superuser

Restrictions

EXPORT_STATISTICS does not export statistics for LONG data type columns.

Examples

The following statement exports statistics on the VMart example database to a file:

```
=> SELECT EXPORT_STATISTICS('/opt/vertica/examples/VMart_Schema/vmart_stats.xml');
      EXPORT_STATISTICS
-----
Statistics exported successfully
(1 row)
```

The next statement exports statistics on a single column (price) from a table named food:

```
=> SELECT EXPORT_STATISTICS('/opt/vertica/examples/VMart_Schema/price.xml', 'food.price');
      EXPORT_STATISTICS
-----
```

```
Statistics exported successfully  
(1 row)
```

See Also

- [ANALYZE_STATISTICS](#)
- [DROP_STATISTICS](#)
- [IMPORT_STATISTICS](#)
- [VALIDATE_STATISTICS](#)
- [Collecting Database Statistics](#)
- [Best Practices For Statistics Collection](#)

IMPORT_STATISTICS

Imports statistics from the XML file that was generated by [EXPORT_STATISTICS](#). Imported statistics override existing statistics for all projections of the table referenced in the XML file.

Syntax

```
IMPORT_STATISTICS ( 'filename' )
```

Parameters

<i>filename</i>	The path and name of an XML input file that was generated by EXPORT_STATISTICS .
-----------------	--

Privileges

Superuser

Restrictions

- `IMPORT_STATISTICS` imports only valid statistics. If the source XML file has invalid statistics for a specific column, those statistics are not imported and Vertica throws a

warning. If the statistics file has an invalid structure, the import operation fails. To check a statistics file for validity, run [VALIDATE_STATISTICS](#).

- `IMPORT_STATISTICS` returns warnings for LONG data type columns, as the source XML file generated by `EXPORT_STATISTICS` contains no statistics for columns of that type.

Example

Import the statistics for the VMart database that `EXPORT_STATISTICS` saved.

```
=> SELECT IMPORT_STATISTICS('/opt/vertica/examples/VMart_Schema/vmart_stats.xml');
      IMPORT_STATISTICS
-----
Importing statistics for projection date_dimension_super column date_key failure (stats did not
contain row counts)
Importing statistics for projection date_dimension_super column date failure (stats did not contain
row counts)
Importing statistics for projection date_dimension_super column full_date_description failure (stats
did not contain row counts)
...
(1 row)
```

See Also

- [ANALYZE_STATISTICS](#)
- [DROP_STATISTICS](#)
- [EXPORT_STATISTICS](#)
- [Collecting Statistics](#) in the Administrator's Guide

VALIDATE_STATISTICS

Validates statistics in the XML file generated by the `EXPORT_STATISTICS` command.

Syntax

```
VALIDATE_STATISTICS ( 'XML_statistics_file' )
```

Parameters

<i>XML_statistics_file</i>	Specifies the path and name of the XML file that contains the statistics you want to validate, type VARCHAR.
----------------------------	--

Privileges

Superuser

Usage Considerations

- [IMPORT_STATISTICS](#) imports only valid statistics. If the source file has invalid statistics for a specific column, those statistics are not imported and a warning occurs. To identify the invalid statistics, run [VALIDATE_STATISTICS](#).

Example: Valid Statistics

The following example shows the results when the statistics are valid:

```
=> SELECT EXPORT_STATISTICS('cust_dim_stats.xml', 'customer_dimension');
EXPORT_STATISTICS
-----
Statistics exported successfully
(1 row)

=> SELECT VALIDATE_STATISTICS('cust_dim_stats.xml');
VALIDATE_STATISTICS
-----
(1 row)
```

Example: Invalid Statistics File

The following example shows the results when some of the statistics are invalid. For example, the 'distinct', 'buckets', 'rows', 'count', and 'distinctCount' attributes cannot be negative numbers. Micro Focus recommends that you rerun [ANALYZE_STATISTICS](#) on this table to create valid statistics:

```
=> SELECT VALIDATE_STATISTICS('/stats.xml');
WARNING 0: Invalid value '-1' for attribute 'distinct' under column 'public.t.x'.
Please use a positive value.
WARNING 0: Invalid value '-1' for attribute 'buckets' under column 'public.t.x'.
Please use a positive value.
```

```
WARNING 0: Invalid value '-1' for attribute 'rows' under column 'public.t.x'.  
Please use a positive value.  
WARNING 0: Invalid value '-1' for attribute 'count' under bound '1', column 'public.t.x'.  
Please use a positive value.  
WARNING 0: Invalid value '-1' for attribute 'distinctCount' under bound '1', column 'public.t.x'.  
Please use a positive value.  
VALIDATE_STATISTICS  
-----  
  
(1 row)
```

Example: Invalid Statistics File

The following example shows the result when the statistics file is invalid:

```
=> SELECT VALIDATE_STATISTICS('/home/dbadmin/stats.xml');  
VALIDATE_STATISTICS  
-----  
Error validating statistics file: At line 1:1. Invalid document structure  
(1 row)
```

See Also

- [ANALYZE_STATISTICS](#)
- [DROP_STATISTICS](#)
- [EXPORT_STATISTICS](#)
- [IMPORT_STATISTICS](#)

Storage Management Functions

This section contains storage management functions specific to Vertica.

ALTER_LOCATION_USE

Alters the type of files that can be stored at the specified storage location.

Syntax

```
ALTER_LOCATION_USE ( 'path' , 'node' , 'usage' )
```

Parameters

<i>path</i>	Specifies where the storage location is mounted.
<i>node</i>	[Optional] The Vertica node with the storage location. Specifying the <i>node</i> parameter as an empty string (' ') alters the location across all cluster nodes in a single transaction.
<i>usage</i>	Is one of the following: <ul style="list-style-type: none">• DATA: The storage location stores only data files. This is the supported use for both a USER storage location, and a labeled storage location.• TEMP: The location stores only temporary files that are created during loads or queries.• DATA,TEMP: The location can store both types of files.

Privileges

Superuser

USER Storage Location Restrictions

You cannot change a storage location from a USER usage type if you created the location that way, or to a USER type if you did not. You can change a USER storage location to specify DATA

(storing TEMP files is not supported). However, doing so does not affect the primary objective of a USER storage location, to be accessible by non-dbadmin users with assigned privileges.

Monitoring Storage Locations

Disk storage information that the database uses on each node appears in the [V_MONITOR.DISK_STORAGE](#) system table.

Example

The following example alters the storage location across all cluster nodes to store only data:

```
=> SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/' , '' , 'DATA');
```

See Also

- [Altering Location Use](#)
- [DROP_LOCATION](#)
- [RESTORE_LOCATION](#)
- [RETIRE_LOCATION](#)
- [GRANT \(Storage Location\)](#)
- [REVOKE \(Storage Location\)](#)

ALTER_LOCATION_LABEL

Alters the location label. Use this function to add, change, or remove a location label. You change a location label only if it is not currently in use as part of a storage policy.

You can use this function to remove a location label. However, you cannot remove a location label if the name being removed is used in a storage policy, *and* the location from which you are removing the label is the last available storage for its associated objects.

Note: If you label an existing storage location that already contains data, and then include the labeled location in one or more storage policies, existing data could be moved. If the ATM determines data stored on a labeled location does not comply with a storage policy, the ATM moves the data elsewhere.

Syntax

```
ALTER_LOCATION_LABEL ( 'path' , 'node' , 'Location_Label' )
```

Parameters

<i>path</i>	Specifies the path of the storage location.
<i>node</i>	The Vertica node for the storage location. If you enter node as an empty string (' '), the function performs a cluster-wide label change to all nodes. Any node that is unavailable generates an error.
<i>Location_Label</i>	Specifies a storage label as a string, for instance <i>SSD</i> . You can change an existing label assigned to a storage location, or add a new label. Specifying an empty string (" ") removes an existing label.

Privileges

Superuser

Example

The following example alters (or adds) the label *SSD* to the storage location at the given path on all cluster nodes:

```
=> SELECT alter_location_label('/home/dbadmin/SSD/tables','', 'SSD');
       alter_location_label
-----
/home/dbadmin/SSD/tables label changed.
(1 row)
```

See Also

- [Altering Location Labels](#)
- [CLEAR_OBJECT_STORAGE_POLICY](#)
- [SET_OBJECT_STORAGE_POLICY](#)

COMPACT_STORAGE

Bundles existing data (.fdb) and index (.pidx) files into the .gt file format. The .gt format was introduced in Vertica 7.2 and is enabled by default for any data files created version 7.2 or later. If you have a database from an earlier version, you can use COMPACT_STORAGE to upgrade existing storage files to the new format. Your database can continue to operate with a mix of file storage formats.

If the settings you specify for COMPACT_STORAGE vary from the limit specified in configuration parameter MaxBundleableROSSizeKB, Vertica does not change the size of the automatically created bundles. You can use this function even if storage bundling is not enabled on your database.

Note: Run this function during periods of low demand.

Syntax

```
SELECT COMPACT_STORAGE (object-name, min-ros-filesize-kb, small-or-all-files, simulate);
```

Parameters

<i>object-name</i>	Specifies the table or projection to bundle.
<i>min-ros-filesize-kb</i>	<p>Specifies the minimum size, in kilobytes, of an independent ROS file. Vertica bundles storage container ROS files below this size into a single file.</p> <p>If set to 0, Vertica bundles the data and index files of an individual column, but not with other columns in that storage container.</p>
<i>small-or-all-files</i>	<p>One of the following:</p> <ul style="list-style-type: none">• <i>small</i>: Bundles only files smaller than the limit specified in <code>min_ros_filesize_kb</code>• <i>all</i>: Bundles files smaller than the limit specified in <code>min_ros_filesize_kb</code> and bundles the .fdb and .pidx files for larger storage containers.
<i>simulate</i>	Specifies whether to simulate the storage settings and

	<p>produce a report describing the impact of those settings.</p> <ul style="list-style-type: none">• <code>true</code>: Produces a report on the impact of the specified bundle settings without actually bundling storage files.• <code>false</code>: Vertica performs the bundling according to the settings you specified.
--	--

Privileges

Superuser

Storage and Performance Impact

Bundling reduces the number of files in your file system by at least fifty percent and improves the performance of file-intensive operations. Improved operations include backups, restores, mergeouts and moveouts.

Vertica creates small files for the following reasons:

- Tables contain hundreds of columns.
- Partition ranges are small (partition by minute).
- Local segmentation is enabled and your factor is set to a high value.

Evaluating the Benefits of Bundled Storage

You can determine whether bundling existing storage can provide a significant benefit, as follows:

View the median file size of a projection on a node

Run the following query:

```
SELECT MEDIAN(size) OVER() AS median_fsize
FROM vs_ros AS ros, storage_containers AS cont
WHERE ros.delid=cont.storage_oid
      AND cont.node_name='node-name'
      AND cont.projection_name='proj-name' LIMIT 1;
```

If many files are smaller than 1mb, bundling storage is likely to provide significant performance benefits.

Run `COMPACT_STORAGE` in simulation mode

The `simulate` parameter produces a report that shows the number of affected files and the change in file storage.

Example

The following example describes the impact of bundling the table `EMPLOYEES`:

```
=> SELECT COMPACT_STORAGE('employees',1024,'small','true');
Task: compact_storage

On node v_vmart_node0001:
Projection Name :public.employees_b0 | selected_storage_containers :0 |
selected_files_to_compact :0 | files_after_compact : 0 | modified_storage_KB :0

On node v_vmart_node0002:
Projection Name :public.employees_b0 | selected_storage_containers :1 |
selected_files_to_compact :6 | files_after_compact : 1 | modified_storage_KB :0

On node v_vmart_node0003:
Projection Name :public.employees_b0 | selected_storage_containers :2 |
selected_files_to_compact :12 | files_after_compact : 2 | modified_storage_KB :0

On node v_vmart_node0001:
Projection Name :public.employees_b1 | selected_storage_containers :2 |
selected_files_to_compact :12 | files_after_compact : 2 | modified_storage_KB :0

On node v_vmart_node0002:
Projection Name :public.employees_b1 | selected_storage_containers :0 |
selected_files_to_compact :0 | files_after_compact : 0 | modified_storage_KB :0

On node v_vmart_node0003:
Projection Name :public.employees_b1 | selected_storage_containers :1 |
selected_files_to_compact :6 | files_after_compact : 1 | modified_storage_KB :0

Success

(1 row)
```

CLEAR_CACHES

Clears the Vertica internal cache files.

Syntax

```
CLEAR_CACHES ( )
```

Privileges

Superuser

Notes

If you want to run benchmark tests for your queries, in addition to clearing the internal Vertica cache files, clear the Linux file system cache. The kernel uses unallocated memory as a cache to hold clean disk blocks. If you are running version 2.6.16 or later of Linux and you have root access, you can clear the kernel filesystem cache as follows:

1. Make sure that all data in the cache is written to disk:

```
# sync
```

2. Writing to the `drop_caches` file causes the kernel to drop clean caches, entries, and inodes from memory, causing that memory to become free, as follows:

- To clear the page cache:

```
# echo 1 > /proc/sys/vm/drop_caches
```

- To clear the entries and inodes:

```
# echo 2 > /proc/sys/vm/drop_caches
```

- To clear the page cache, entries, and inodes:

```
# echo 3 > /proc/sys/vm/drop_caches
```

Example

The following example clears the Vertica internal cache files:

```
=> SELECT CLEAR_CACHES();
CLEAR_CACHES
-----
Cleared
(1 row)
```

CLEAR_OBJECT_STORAGE_POLICY

Removes an existing storage policy. The specified object will no longer use a user-created storage location. Any existing data stored currently at the labeled location in the object's storage policy is moved to default storage during the next TM moveout operation.

Syntax

```
CLEAR_OBJECT_STORAGE_POLICY ( 'object_name' [, 'key_min', 'key_max'] [, 'enforce_storage_move' ] )
```

Parameters

<i>object_name</i>	Identifies the database object whose storage policies are to be cleared. The <i>object_name</i> parameter can resolve to a database, schema, or table.
<i>key_min, key_max</i>	[Optional] Specifies the table partition key value ranges stored at the labeled location. These parameters are applicable only when <i>object_name</i> is a table.
<i>enforce_storage_move</i>	[Optional] If true, moves the storage containers that belong to this object to the new location immediately. If false (the default), this move occurs automatically sometime after the next moveout operation.

Privileges

Superuser

Examples

This example shows how to clear the storage policy for the object `lineorder`. Changes take effect after the next moveout:

```
=> select clear_object_storage_policy('lineorder');
       clear_object_storage_policy
-----
Default storage policy cleared.
(1 row)
```

See Also

- [Clearing Storage Policies](#)
- [ALTER_LOCATION_LABEL](#)

- [SET_OBJECT_STORAGE_POLICY](#)
- [ENFORCE_OBJECT_STORAGE_POLICY](#)

DROP_LOCATION

Removes the specified storage location. Dropping a storage location is a permanent operation and cannot be undone. Therefore, you must [retire a storage location](#) before dropping it.

Syntax

```
DROP_LOCATION ( 'path' , 'node' )
```

Parameters

<i>path</i>	Specifies where the storage location to drop is mounted.
<i>node</i>	The Vertica node where the location is available. A value of "" (an empty string) means to perform this operation on all nodes.

Privileges

Superuser

Retiring a Storage Location

Retiring a storage location lets you verify that you do not need the storage before dropping it. You can also restore a retired storage location if you determine it is still in use.

Dropping a Shared Location

When a storage location is created in a shared location, such as HDFS, Vertica creates subdirectories for each node to prevent conflicts. For example, suppose you create a shared storage location in /data. The location for v_vmart_node0001 is /data/v_vmart_node0001, the location for v_vmart_node0002 is /data/v_vmart_node0002, and so on. To drop a location on only one node, use the node-specific path such as /data/v_vmart_node0002. To drop a location on all nodes, use the same path that you used to create it (/data in this example).

Storage Locations with Temp and Data Files

If you use a storage location to store data and then alter it to store only temp files, the location can still contain data files. Vertica does not let you drop a storage location containing data files. You can use the function [MOVE_RETIRED_LOCATION_DATA](#) to manually merge out the data files from the storage location, or you can drop partitions. Deleting data files does not work.

Examples

The following example shows how to drop a previously retired storage location on `v_vmart_node0003`:

```
=> SELECT DROP_LOCATION('/data', 'v_vmart_node0003');
```

See Also

- [Dropping Storage Locations](#)
- [Retiring Storage Locations](#)
- [ALTER_LOCATION_USE](#)
- [RESTORE_LOCATION](#)
- [RETIRE_LOCATION](#)
- [GRANT \(Storage Location\)](#)
- [REVOKE \(Storage Location\)](#)

ENFORCE_OBJECT_STORAGE_POLICY

Applies object storage policies immediately, instead of waiting for the Tuple Mover to perform the next moveout. Calling this function is equivalent to setting the `enforce_storage_move` parameter on related meta-functions. You typically use this function as the last step before dropping a storage location.

This function invokes the Tuple Mover on a one-time basis.

Syntax

```
ENFORCE_OBJECT_STORAGE_POLICY ( 'object_name', [, 'key_min', 'key_max'] )
```

Parameters

<i>object_name</i>	Identifies the database object whose storage policies are to be applied. The <i>object_name</i> parameter can resolve to a database, schema, or table.
<i>key_min</i> , <i>key_max</i>	Applicable only when <i>object_name</i> is a table, <i>key_min</i> and <i>key_max</i> specify the table partition key value range over which to perform the moves.

Privileges

Must be the object owner to enforce the storage policy and have access to the storage location.

Examples

This example shows how to apply storage-policy updates to the test table:

```
=> select enforce_object_storage_policy("test");
```

See Also

- [CLEAR_OBJECT_STORAGE_POLICY](#)
- [RETIRE_LOCATION](#)
- [DROP_LOCATION](#)
- [Managing Storage Locations](#)

MEASURE_LOCATION_PERFORMANCE

Measures disk performance for the location specified.

Syntax

```
MEASURE_LOCATION_PERFORMANCE ( 'path' , 'node' )
```

Parameters

<i>path</i>	Specifies where the storage location to measure is mounted.
<i>node</i>	Is the Vertica node where the location to be measured is available.

Privileges

Superuser

Notes

- To get a list of all node names on your cluster, query the [V_MONITOR.DISK_STORAGE](#) system table:

```
=> SELECT node_name from DISK_STORAGE;
      node_name
-----
v_vmartdb_node0004
v_vmartdb_node0004
v_vmartdb_node0005
v_vmartdb_node0005
v_vmartdb_node0006
v_vmartdb_node0006
(6 rows)
```

- If you intend to create a tiered disk architecture in which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns, you need to measure storage location performance for each location in which data is stored. You do not need to measure storage location performance for temp data storage locations because temporary files are stored based on available space.
- The method of measuring storage location performance applies only to configured clusters. If you want to measure a disk before configuring a cluster see [Measuring Storage Performance](#).
- Storage location performance equates to the amount of time it takes to read and write 1MB of data from the disk. This time equates to:

IO time = Time to read/write 1MB + Time to seek = 1/Throughput + 1/Latency

Throughput is the average throughput of sequential reads/writes (units in MB per second)

Latency is for random reads only in seeks (units in seeks per second)

Note: The IO time of a faster storage location is less than a slower storage location.

Example

The following example measures the performance of a storage location on v_vmartdb_node0004:

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/' , 'v_vmartdb_node0004');  
WARNING: measure_location_performance can take a long time. Please check logs for progress  
measure_location_performance  
-----  
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

See Also

- [CREATE LOCATION](#)
- [ALTER_LOCATION_USE](#)
- [RESTORE_LOCATION](#)
- [RETIRE_LOCATION](#)
- [Measuring Storage Performance](#)

MOVE_RETIRED_LOCATION_DATA

Moves all data from either a single retired storage location or all retired storage locations in the database. This function migrates the data to non-retired storage locations based on the storage policies of the objects whose data is stored in the location.

Syntax

```
MOVE_RETIRED_LOCATION_DATA(['location_path'] ['node'])
```

Arguments

['location_path']	The path of the storage location as listed in the LOCATION_PATH column
-------------------	--

	<p>of the STORAGE_LOCATIONS system table. You must have previously marked this storage location as retired.</p> <p>Default Value: If you do not supply a storage location path, this function moves data from all retired storage locations.</p>
['node']	<p>A specific node on which to move the retired storage location's data.</p> <p>Default Value: If you do not specify a node, all nodes in the cluster move data from the retired location or locations. If a node does not define the storage location, this function returns an error.</p>

Privileges

The user must be a superuser .

Usage Considerations

This function forces the Tuple Mover to move data out of the retired location or locations. Normally, the Tuple Mover only migrates data out of retired storage locations as it consolidates data into larger ROS containers. This function does not return until all of the data has moved off of the retired storage location or locations.

Examples

The following example:

1. Queries the `STORAGE_LOCATIONS` system table to show which storage locations are retired.
2. Queries the `STORAGE_CONTAINERS` system table to show the current location of the messages table, which is currently stored in the retired storage location named `ssd`.
3. Calls `MOVE_RETIRED_LOCATION_DATA` to move the data off of the `ssd` storage location.
4. Repeats the previous query to show the storage location of the messages table.

```
=> SELECT node_name, location_path, location_label, is_retired FROM STORAGE_LOCATIONS
    WHERE is_retired = 't';
   node_name      | location_path      | location_label | is_retired
-----+-----+-----+-----
v_vmart_node0001 | /home/dbadmin/SSDLoc | ssd           | t
v_vmart_node0002 | /home/dbadmin/SSDLoc | ssd           | t
v_vmart_node0003 | /home/dbadmin/SSDLoc | ssd           | t
```

(3 rows)

```
=> SELECT node_name, total_row_count, storage_type, location_label FROM STORAGE_CONTAINERS  
WHERE projection_name ILIKE 'messages%';
```

node_name	total_row_count	storage_type	location_label
v_vmart_node0001	333514	ROS	ssd
v_vmart_node0001	333255	ROS	ssd
v_vmart_node0002	333255	ROS	ssd
v_vmart_node0002	333231	ROS	ssd
v_vmart_node0003	333231	ROS	ssd
v_vmart_node0003	333514	ROS	ssd

(6 rows)

```
=> SELECT MOVE_RETIRED_LOCATION_DATA('/home/dbadmin/SSDLoc');  
MOVE_RETIRED_LOCATION_DATA
```

Move data off retired storage locations done

(1 row)

```
=> SELECT node_name, total_row_count, storage_type, location_label FROM storage_containers  
WHERE projection_name ILIKE 'messages%';
```

node_name	total_row_count	storage_type	location_label
v_vmart_node0001	333255	ROS	base
v_vmart_node0001	333514	ROS	base
v_vmart_node0003	333514	ROS	base
v_vmart_node0003	333231	ROS	base
v_vmart_node0002	333231	ROS	base
v_vmart_node0002	333255	ROS	base

(6 rows)

See Also

- [RETIRE_LOCATION](#)
- [RESTORE_LOCATION](#)
- [Managing Storage Locations](#) in the Administrator's Guide.

RESTORE_LOCATION

Restores a storage location that was previously retired with [RETIRE_LOCATION](#).

Syntax

```
RESTORE_LOCATION ( 'path', 'node' )
```

Parameters

<i>path</i>	Specifies where the retired storage location is mounted.
<i>node</i>	The Vertica node where the retired location is available. A value of "" (an empty string) means to perform this operation on all nodes. The operation fails if you have dropped any locations.

Privileges

Superuser

Effects of Restoring a Previously Retired Location

After restoring a storage location, Vertica re-ranks all of the cluster storage locations. It uses the newly restored location to process queries as determined by its rank.

Monitoring Storage Locations

Disk storage information that the database uses on each node appears in the [V_MONITOR.DISK_STORAGE](#) system table.

Examples

The following example shows how to restore the retired storage location on node3:

```
=> SELECT RESTORE_LOCATION ('/thirdVerticaStorageLocation/' , 'v_vmartdb_node004');
```

See Also

- [Altering Location Use](#)
- [CREATE LOCATION](#)
- [ALTER_LOCATION_USE](#)
- [DROP_LOCATION](#)
- [RETIRE_LOCATION](#)

- [GRANT \(Storage Location\)](#)
- [REVOKE \(Storage Location\)](#)

RETIRE_LOCATION

Makes the specified storage location inactive. To see a list of all of the storage locations in Vertica, refer to [STORAGE_LOCATIONS](#).

Syntax

```
RETIRE_LOCATION ( 'path', 'node' [, enforce_storage_move ] )
```

Parameters

<i>path</i>	Specifies where the storage location to retire resides.
<i>node</i>	The Vertica node where the location is available. A value of "" (an empty string) means to perform this operation on all nodes.
<i>enforce_storage_move</i>	[Optional] Use this to expedite dropping a location. If you set this argument to true, the location label is set to the empty string, and the data is moved elsewhere. The location can then be dropped without errors or warnings.

Privileges

Superuser

Retiring a Shared Location

When you create a storage location in a shared HDFS location, Vertica creates subdirectories for each node to prevent conflicts. For example, suppose you create a shared storage location in /data. The location for v_vmartdb_node0001 is /data/v_vmartdb_node0001, the location for v_vmartdb_node0002 is /data/v_vmartdb_node0002, and so on. To retire a location on only one node, use the node-specific path such as /data/v_vmartdb_node0002. To retire a location on all nodes, use the same path that you used to create it (/data in this example).

Note: Vertica does not identify NFS as a shared file system.

Effects of Retiring a Storage Location

When you use this function, Vertica checks that the location is not the only storage for data and temp files. At least one location must exist on each node to store data and temp files. However, you can store both sorts of files in either the same location or separate locations.

Note: If a location is the last available storage for its associated objects, you cannot retire it *unless* you set `enforce_storage_move` to `true`

When you retire a storage location:

- No new data is stored at the retired location, unless you first restore it with the [RESTORE_LOCATION\(\)](#) function.
- By default, if the storage location being retired contains stored data, the data is not moved. Thus, you cannot drop the storage location. Instead, Vertica removes the stored data through one or more mergeouts. If you want to drop the location immediately after retiring it, add the `enforce_storage_move` parameter, using a value of `true`.
- If the storage location being retired is used only for temp files or you use `enforce_storage_move`, you can drop the location. See [Dropping Storage Locations](#) in the Administrators Guide and the [DROP_LOCATION\(\)](#) function.

Monitoring Storage Locations

Disk storage information that the database uses on each node appears in the [V_MONITOR.DISK_STORAGE](#) system table.

Examples

The following examples show two approaches to retiring a storage location.

You can specify that a storage location be dropped automatically at a future time:

```
=> SELECT RETIRE_LOCATION ('/data' , 'v_vmartdb_node0004');
```

You can also specify that a storage location be dropped immediately:

```
=> SELECT RETIRE_LOCATION ('/data' , 'v_vmartdb_node0004', true);
```

See Also

- [Retiring Storage Locations](#)
- [CREATE LOCATION](#)
- [ALTER_LOCATION_USE](#)
- [DROP_LOCATION](#)
- [RESTORE_LOCATION](#)
- [GRANT \(Storage Location\)](#)
- [REVOKE \(Storage Location\)](#)

SET_LOCATION_PERFORMANCE

Sets disk performance for the location specified.

Syntax

```
SET_LOCATION_PERFORMANCE ('path', 'node' , 'throughput' , 'average_latency')
```

Parameters

<i>path</i>	Specifies where the storage location to set is mounted.
<i>node</i>	Is the Vertica node where the location to be set is available. If this parameter is omitted, <i>node</i> defaults to the initiator.
<i>throughput</i>	Specifies the throughput for the location, which must be 1 or more.
<i>average_latency</i>	Specifies the average latency for the location. The <i>average_latency</i> must be 1 or more.

Privileges

Superuser

Notes

To obtain the throughput and average latency for the location, run the [MEASURE_LOCATION_PERFORMANCE\(\)](#) function before you attempt to set the location's performance.

Example

The following example sets the performance of a storage location on node2 to a throughput of 122 megabytes per second and a latency of 140 seeks per second.

```
=> SELECT SET_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/', 'node2', '122', '140');
```

See Also

- [CREATE LOCATION](#)
- [MEASURE_LOCATION_PERFORMANCE](#)
- [Measuring Storage Performance](#)
- [Setting Storage Performance](#)

SET_OBJECT_STORAGE_POLICY

Creates or changes an object storage policy by associating a database object with a labeled storage location.

Note: You cannot create a storage policy on a USER type storage location.

Syntax

```
SET_OBJECT_STORAGE_POLICY ( 'object_name', 'location_label'  
    [, 'key_min', 'key_max'] [, enforce_storage_move] )
```

Parameters

<i>object_name</i>	Identifies the database object assigned to a labeled storage location. The <i>object_name</i> can resolve to a database, schema, or table.
<i>location_label</i>	The label of the storage location with which <i>object_name</i> is being

	associated.
<i>key_min, key_max</i>	[Optional] Applicable only when <i>object_name</i> is a table, <i>key_min</i> and <i>key_max</i> specify the table partition key value range to be stored at the location.
<i>enforce_storage_move</i>	[Optional] Specify this parameter as <code>true</code> to move all existing storage data to the target location within this function's transaction. By default, data move the next time the Tuple Mover runs on the affected data.

Privileges

Must be the object owner to set the storage policy and have access to the storage location.

New Storage Policy

If an object does not have a storage policy, this function creates a new policy. The labeled location is then used as the default storage location during TM operations, such as moveout and mergeout.

Existing Storage Policy

If the object already has an active storage policy, calling this function changes the default storage for the object to the new labeled location. Any existing data stored on the previous storage location is marked to move to the new location during the next TM moveout operations. To move the data immediately, use the *enforce_storage_move* argument.

Forcing Existing Data Storage to a New Storage Location

You can optionally use this function to move existing data storage to a new location as part of completing the current transaction. To do so, specify the last parameter as `true`. You might want to force a move, even though it means waiting for the operation to complete before continuing, if the data being moved is old. The Tuple Mover runs less frequently on older data.

To move existing data as part of the next TM moveout, either omit the parameter, or specify its value as `false`.

Note: Specifying the parameter as `true` performs a cluster-wide operation. If an error occurs on any node, the function displays a warning message and skips the node where the error occurs. It then continues executing the operation on the remaining nodes.

Examples

This example shows how to set a storage policy for the table `test` to use the storage labeled SSD as its default location:

```
=> select set_object_storage_policy('test','ssd', true);
set_object_storage_policy
-----
Object storage policy set.
Task: moving storages
(Table: public.test) (Projection: public.test_b0)
(Table: public.test) (Projection: public.test_b1)

(1 row)
```

See Also

- [ALTER_LOCATION_LABEL](#)
- [CLEAR_OBJECT_STORAGE_POLICY](#)
- [Creating Storage Policies](#)
- [ENFORCE_OBJECT_STORAGE_POLICY](#)
- [Moving Data Storage Locations](#)

Text Search Functions

This section contains text search functions specific to Vertica.

DELETE_TOKENIZER_CONFIG_FILE

Deletes a tokenizer configuration file.

Syntax

```
SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE (USING PARAMETERS proc_oid='proc_oid', confirm={true | false });
```

Parameters

<code>confirm = [true false]</code>	<p>Boolean flag. Indicates that the configuration file should be removed even if the tokenizer is still in use.</p> <p>True — Force deletion of the tokenizer when the used parameter value is True.</p> <p>False — Delete tokenizer if the used parameter value is False.</p> <p>Default: False</p>
<code>proc_oid</code>	<p>A unique identifier assigned to a tokenizer when it is created. Users must query the system table <code>vs_procedures</code> to get the <code>proc_oid</code> for a given tokenizer name. See Configuring a Tokenizer for more information.</p>

Examples

The following example shows how you can use `DELETE_TOKENIZER_CONFIG_FILE` to delete the tokenizer configuration file:

```
=> SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE (USING PARAMETERS proc_oid='45035996274126984');
DELETE_TOKENIZER_CONFIG_FILE
-----
t
(1 row)
```

GET_TOKENIZER_PARAMETER

Returns the configuration parameter for a given tokenizer.

Syntax

```
SELECT v_txtindex.GET_TOKENIZER_PARAMETER(parameter_name USING PARAMETERS proc_oid='proc_oid');
```

Parameters

<code>parameter_name</code>	<p>Name of the parameter to be returned.</p> <p>One of the following:</p>
-----------------------------	---

	<ul style="list-style-type: none">• stopWordsCaseInsensitive• minorSeparators• majorSeparators• minLength• maxLength• ngramsSize• used
proc_oid	A unique identifier assigned to a tokenizer when it is created. Users must query the system table vs_procedures to get the proc_oid for a given tokenizer name. See Configuring a Tokenizer for more information.

Examples

The following examples show how you can use GET_TOKENIZER_PARAMETER.

Return the stop words used in a tokenizer:

```
=> SELECT v_txtindex.GET_TOKENIZER_PARAMETER('stopwordscaseinsensitive' USING PARAMETERS proc_
oid='45035996274126984');
  getTokenizerParameter
-----
devil,TODAY,the,fox
(1 row)
```

Return the major separators used in a tokenizer:

```
=> SELECT v_txtindex.GET_TOKENIZER_PARAMETER('majorseparators' USING PARAMETERS proc_
oid='45035996274126984');
  getTokenizerParameter
-----
{}()&[]
(1 row)
```

READ_CONFIG_FILE

Reads and returns the key-value pairs of all the parameters of a given tokenizer.

You must use the OVER() clause with this function.

Syntax

```
SELECT v_txtindex.READ_CONFIG_FILE(USING PARAMETERS proc_oid='proc_oid') OVER ()
```

Parameters

proc_oid	A unique identifier assigned to a tokenizer when it is created. Users must query the system table vs_procedures to get the proc_oid for a given tokenizer name. See Configuring a Tokenizer for more information.
----------	---

Examples

The following example shows how you can use READ_CONFIG_FILE to return the parameters associated with a tokenizer:

```
=> SELECT v_txtindex.READ_CONFIG_FILE(USING PARAMETERS proc_oid='45035996274126984') OVER();
      config_key | config_value
-----+-----
majorseparators | {}()&[]
stopwordscaseinsensitive | devil,TODAY,the,fox
(2 rows)
```

SET_TOKENIZER_PARAMETER

Configures the tokenizer parameters.

Important: \n, \t, \r must be entered as Unicode using Vertica notation, U&' \000D', or using Vertica escaping notation, E'\r'. Otherwise, they are taken literally as two separate characters. For example, "\" & "r".

Syntax

```
SELECT v_txtindex.SET_TOKENIZER_PARAMETER (parameter_name, parameter_value USING PARAMETERS proc_oid='proc_oid')
```

Parameters

parameter_name	Name of the parameter to be configured. Use one of the following:
----------------	--

- `stopwordsCaseInsensitive` — List of stop words. All the tokens that belong to the list are ignored. Vertica supports separators and stop words up to the first 256 Unicode characters.

If you want to define a stop word that contains a comma or a backslash, then it needs to be escaped.

For example: "Dear Jack\," "Dear Jack\\"

Default: ' ' (empty list)

- `majorSeparators` — List of major separators. Enclose in quotes with no spaces between.

Default: E' []<>(){}|!;, ' '"*&?+\r\n\t'

- `minorSeparators` — List of minor separators. Enclose in quotes with no spaces between.

Default: E' / : = @ . - \$ # % \ _ '

- `minLength` — Minimum length a token can have, type Integer. Must be greater than 0.

Default: ' 2 '

- `maxLength` — Maximum length a token can be. Type Integer. Cannot be greater than 1024 bytes. For information about increasing the token size, see [Text Search Parameters](#).

Default: ' 128 '

- `ngramsSize` — Integer value greater than zero. Use only with ngram tokenizers.

Default: ' 3 '

- `used` — Indicates when a tokenizer configuration cannot be changed. Type Boolean. After you set `used` to True, any calls to `setTokenizerParameter` fail.

You must set the parameter `used` to True before using the configured tokenizer. Doing so prevents the configuration from being modified after being used to create a text index.

Default: False

parameter_value	The value of a configuration parameter. If you want to disable minorSeparators or stopWordsCaseInsensitive, then set their values to ' '.
proc_oid	A unique identifier assigned to a tokenizer when it is created. Users must query the system table vs_procedures to get the proc_oid for a given tokenizer name. See Configuring a Tokenizer for more information.

Examples

The following examples show how you can use SET_TOKENIZER_PARAMETER to configure stop words and separators.

Configure the stop words of a tokenizer:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('stopwordsCaseInsensitive', 'devil,TODAY,the,fox' USING
PARAMETERS proc_oid='45035996274126984');
SET_TOKENIZER_PARAMETER
-----
t
(1 row)
```

Configure the major separators of a tokenizer:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('majorSeparators',E'{}()&[]' USING PARAMETERS proc_
oid='45035996274126984');
SET_TOKENIZER_PARAMETER
-----
t
(1 row)
```

Table Management Functions

This section contains the functions associated with the Vertica library table management.

COPY_TABLE

Copies one table to another. This lightweight, in-memory copy increases performance by initially sharing the same storage between two tables. The copied table includes copies of any explicitly created projections from the source table. Once copied, the source and copy tables

are independent of each other. Users can perform operations on one table without impacting the other. These operations can increase the overall storage required for both tables.

Creating multiple, concurrent copies of the same table may cause some of the copies to fail. To ensure success, copy tables sequentially.

Note: Although they share storage space, Vertica considers the tables as discrete objects for license capacity purposes. For example, copying a one TB table would only consume one TB of space. Your Vertica license, however, considers them as separate objects consuming two TB of space.

Syntax

```
COPY_TABLE (  
  '[schema.]source-table',  
  '[schema.]target-table'  
)
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDbObject</pre>
<i>source-table</i>	<p>The source table to copy. Vertica copies all data from this table to the target table.</p>
<i>target-table</i>	<p>The target table of the source table. If the target table already exists, Vertica appends the source to the existing table.</p> <p>If the table does not exist, Vertica creates a table from the source table's definition, by calling CREATE TABLE with LIKE and INCLUDING PROJECTIONS clause. The new table inherits ownership from the source table. For details, see Replicating a Table.</p>

Privileges

- User must have INSERT, UPDATE, DELETE and SELECT privileges on the source table.
- User must have CREATE privileges on the target table and target table schema if COPY_TABLE is creating a new table.

- User must have INSERT privileges on the target table if COPY_TABLE is adding to an existing table.

Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partitioning expression
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled.

Note: If the target table has primary or unique key constraints enabled and moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation. Enforcing constraints requires disk reads and can slow the copy process.

- Number and definitions of text indices.

Table Restrictions

The following restrictions apply to the source and target tables:

- If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.
- If the source table contains a sequence, Vertica converts the sequence to an integer before copying it to the target table. If the target table contains auto-increment, identity, or named sequence columns, Vertica cancels the copy and displays an error message.
- The following tables cannot be used as sources or targets:
 - Temporary tables
 - Virtual tables

- System tables
- External tables

Examples

If you call `COPY_TABLE` and the target table does not exist, the function creates the table automatically. In the following example, the target table `public.newtable` does not exist. `COPY_TABLE` creates the table and replicates the source table. Vertica also copies all the constraints associated with the source table except foreign key constraints.

```
=> SELECT COPY_TABLE (  
      'public.product_dimension',  
      'public.newtable');  
-[ RECORD 1 ]-----  
copy_table | Created table public.newtable.  
Copied table public.product_dimension to public.newtable
```

REBALANCE_TABLE

Synchronously rebalances data in the specified table.

A rebalance operation performs the following tasks:

- Distributes data based on:
 - User-defined [fault groups](#), if specified
 - [Large cluster](#) automatic fault groups
- Redistributes database projection data across all nodes.

Syntax

```
REBALANCE_TABLE(' [schema.] table-name')
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
---------------	---

<i>table-name</i>	The table to rebalance.
-------------------	-------------------------

Privileges

Superuser

When to Rebalance

Rebalancing is useful or even necessary after you perform the following tasks:

- Mark one or more nodes as ephemeral in preparation of removing them from the cluster.
- Add one or more nodes to the cluster so that Vertica can populate the empty nodes with data.
- Change the [scaling factor](#) of an elastic cluster, which determines the number of storage containers used to store a projection across the database.
- Set the control node size or realign control nodes on a [large cluster](#) layout
- Add nodes to or remove nodes from a [fault group](#).

Tip: By default, before performing a rebalance, Vertica queries system tables to compute the size of all projections involved in the rebalance task. This query can add significant overhead to the rebalance operation. To disable this query, set projection configuration parameter [RebalanceQueryStorageContainers](#) to 0.

Example

The following command shows how to rebalance data on the specified table.

```
=> SELECT REBALANCE_TABLE('online_sales.online_sales_fact');
REBALANCE_TABLE
-----
REBALANCED
(1 row)
```

See Also

- [REBALANCE_CLUSTER](#)
- [Rebalancing Data Across Nodes](#)

- [NODES](#)

Tuple Mover Functions

This section contains tuple mover functions specific to Vertica.

DO_TM_TASK

Runs a Tuple Mover operation on the specified table or projection and commits any current transaction.

Tip: Running this function does not require you to stop the tuple mover.

Syntax

```
DO_TM_TASK('task'[, 'schema.]{table | projection}')
```

Parameters

<i>task</i>	<p>Specifies one of the following tuple mover operations:</p> <ul style="list-style-type: none">• moveout: Moves out data from WOS to ROS. For details, see Moveout in the Administrator's Guide.• mergeout: Consolidates ROS containers and purges deleted records. For details, see Mergeout in the Administrator's Guide.• analyze_row_count: Collects the number of rows in the specified projection. If you specify a table name, DO_TM_TASK returns the row counts for all projections of that table. <p>Vertica automatically analyzes all projection row counts at the time intervals specified by configuration parameter AnalyzeRowCountInterval.</p> <p>DO_TM_TASK aggregates row counts calculated during loads. It commits this data to the catalog when the percentage of WOS to ROS equals the setting in configuration parameter ARCommitPercentage.</p>
-------------	---

	<p><code>analyze_row_count</code> analyzes the row count of Vertica projections. To calculate row counts for external tables, use ANALYZE_EXTERNAL_ROW_COUNT.</p>
<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table projection</i>	<p>Applies <i>task</i> to the specified table or projection. If you specify a projection and it is not found, <code>DO_TM_TASK</code> looks for a table with that name and, if found, applies the task to it and all projections associated with it.</p> <p>If no table or projection is specified, the task is applied to all tables and projections in the database.</p>

Privileges

- Any INSERT/UPDATE/DELETE privilege on table
- USAGE privileges on schema

Examples

The following example performs a moveout of all projections for the `t1` table:

```
=> SELECT DO_TM_TASK('moveout', 't1');
```

The following example performs a moveout for the `t1_proj` projection :

```
=> SELECT DO_TM_TASK('moveout', 't1_proj');
```

Workload Management Functions

This section contains workload management functions specific to Vertica.

ANALYZE_WORKLOAD

Runs the Workload Analyzer (WLA), a utility that analyzes system information held in [system tables](#).

WLA intelligently monitors the performance of SQL queries and workload history, resources, and configurations to identify the root causes for poor query performance. ANALYZE_WORKLOAD returns tuning recommendations for all events within the scope and time that you specify, from system table [TUNING_RECOMMENDATIONS](#).

Tuning recommendations are based on a combination of [statistics](#), system and data collector events, and database-table-projection design. WLA recommendations can help you quickly and easily tune query performance.

See [Understanding WLA Triggering Conditions](#) in the Administrator's Guide for the common triggering conditions and recommendations.

Syntax

```
ANALYZE_WORKLOAD ( '[ scope ]' [, 'since-time' | save-data ] );
```

Parameters

<i>scope</i>	<p>Specifies the catalog objects to analyze, as follows:</p> <p><i>[schema.]table</i></p> <p>If set to an empty string, Vertica returns recommendations for all database objects.</p>
<i>since-time</i>	<p>Specifies the start time for the analysis time span, which continues up to the current system status, inclusive. If you omit this parameter, ANALYZE_WORKLOAD returns recommendations on events since the last time you called this function.</p> <p>Note: You must explicitly cast strings to <code>TIMESTAMP</code> or <code>TIMESTAMPTZ</code>. For example:</p>

	<pre>SELECT ANALYZE_WORKLOAD('T1', '2010-10-04 11:18:15'::TIMESTAMPZ); SELECT ANALYZE_WORKLOAD('T1', TIMESTAMPZ '2010-10-04 11:18:15');</pre>
<i>save-data</i>	<p>Specifies whether to save returned values from ANALYZE_WORKLOAD:</p> <ul style="list-style-type: none">• <code>false</code> (default): Results are discarded.• <code>true</code>: Saves the results returned by ANALYZE_WORKLOAD. Subsequent calls to ANALYZE_WORKLOAD return results that start from the last invocation when results were saved. Object events preceding that invocation are ignored.

Return Values

Returns aggregated tuning recommendations from [TUNING_RECOMMENDATIONS](#).

Privileges

Superuser

Examples

See [Getting Tuning Recommendations](#) in the Administrator's Guide.

See Also

- [Analyzing Workloads](#)
- [Understanding WLA Triggering Conditions](#)

CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY

Changes the run-time priority of an active query.

Note: This function replaces deprecated function CHANGE_RUNTIME_PRIORITY.

Syntax

```
CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY(transaction-id, 'value')
```

Parameters

<i>transaction-id</i>	Identifies the transaction, obtained from the system table SESSIONS .
<i>value</i>	The RUNTIMEPRIORITY value: HIGH, MEDIUM, or LOW.

Privileges

- Superuser: None
- Non-superusers can only change the runtime priority of their own queries, and cannot raise the runtime priority of a query to a level higher than that of the resource pool.

Example

See [Changing Runtime Priority of a Running Query](#).

CHANGE_RUNTIME_PRIORITY

Changes the run-time priority of a query that is actively running. Note that, while this function is still valid, you should instead use `CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY` to change run-time priority. `CHANGE_RUNTIME_PRIORITY` will be deprecated in a future release of Vertica.

Syntax

```
CHANGE_RUNTIME_PRIORITY(TRANSACTION_ID, STATEMENT_ID, 'value')
```

Parameters

TRANSACTION_ID	An identifier for the transaction within the session. TRANSACTION_ID cannot be NULL. You can find the transaction ID in the Sessions table.
STATEMENT_ID	A unique numeric ID assigned by the Vertica catalog, which identifies the currently executing statement.

	<p>You can find the statement ID in the Sessions table.</p> <p>You can specify NULL to change the run-time priority of the currently running query within the transaction.</p>
<i>'value'</i>	The RUNTIMEPRIORITY value. Can be HIGH, MEDIUM, or LOW.

Privileges

No special privileges required. However, non-superusers can change the run-time priority of their own queries only. In addition, non-superusers can never raise the run-time priority of a query to a level higher than that of the resource pool.

Example

```
=> SELECT CHANGE_RUNTIME_PRIORITY(45035996273705748, NULL, 'low');
```

MOVE_STATEMENT_TO_RESOURCE_POOL

Attempts to move the specified query to the specified target pool.

Syntax

```
MOVE_STATEMENT_TO_RESOURCE_POOL (session_id , transaction_id, statement_id, target_resource_pool_name)
```

Parameters

<i>session_id</i>	Identifier for the session where the query you want to move is currently executing.
<i>transaction_id</i>	Identifier for the transaction within the session.
<i>statement_id</i>	Unique numeric ID for the statement you want to move.
<i>target_resource_pool_name</i>	Name of the existing resource pool to which you want to move the specified query.

Outputs

The function may return the following results:

MOV_REPLAN: Target pool does not have sufficient resources. See `v_monitor.resource_pool_move` for details. Vertica will attempt to replan the statement on target pool.

MOV_REPLAN: Target pool has priority HOLD. Vertica will attempt to replan the statement on target pool.

MOV_FAILED: Statement not found.

MOV_NO_OP: Statement already on target pool.

MOV_REPLAN: Statement is in queue. Vertica will attempt to replan the statement on target pool.

MOV_SUCC: Statement successfully moved to target pool.

Privileges

Superuser

Examples

The following example shows how you can move a specific statement to a resource pool called `my_target_pool`:

```
=> SELECT MOVE_STATEMENT_TO_RESOURCE_POOL ('v_vmart_node0001.example.-31427:0x82fbm',  
45035996273711993, 1, 'my_target_pool');
```

See Also:

- [Manually Moving Queries to Different Resource Pools](#)
- [RESOURCE_POOL_MOVE](#)

SLEEP

Waits a specified number of seconds before executing another statement or command.

Syntax

```
SLEEP( seconds )
```

Parameters

<i>seconds</i>	The wait time, specified in one or more seconds (0 or higher) expressed as a positive integer. Single quotes are optional; for example, <code>SLEEP(3)</code> is the same as <code>SLEEP('3')</code> .
----------------	--

Notes

- This function returns value 0 when successful; otherwise it returns an error message due to syntax errors.
- You cannot cancel a sleep operation.
- Be cautious when using `SLEEP()` in an environment with shared resources, such as in combination with transactions that take exclusive locks.

Example

The following command suspends execution for 100 seconds:

```
=> SELECT SLEEP(100);
sleep
-----
      0
(1 row)
```

SQL Statements

The primary structure of a SQL query is its statement. Multiple statements are separated by semicolons. The following example contains four common SQL statements—CREATE TABLE, INSERT, SELECT, and COMMIT:

```
=> CREATE TABLE comments (id INT, comment VARCHAR);
CREATE TABLE
=> INSERT INTO comments VALUES (1, 'Hello World');
OUTPUT
-----
1
(1 row)

=> SELECT * FROM comments;
id | comment
----+-----
1 | Hello World
(1 row)

=> COMMIT;
COMMIT
=>
```

ALTER ACCESS POLICY

The ALTER ACCESS POLICY statement:

- Enables and disables individual access policies in a table.
- Copies an access policy from one table to another.

Important: When you copy or rename a table, access policies associated with the original table are not included in the copied table. You must use ALTER ACCESS POLICY to copy the access policies to the new table.

Syntax

```
ALTER ACCESS POLICY ON tablename
.. .FOR COLUMN columnname
... expression
... [ENABLE | DISABLE];
... | FOR COLUMN columnname
... COPY TO TABLE tablename;
```

```
ALTER ACCESS POLICY ON tablename  
... FOR ROWS  
...expression  
... [ENABLE | DISABLE];  
...FOR ROWS  
...expression  
... COPY TO TABLE tablename;
```

Parameters

tablename	The name of the table that contains the access policy you want to enable, disable, or copy.
columnname	Either of the following values: <ul style="list-style-type: none">• The name of the column on which to enable or disable the access policy• The name of the column you want to copy
expression	An expression that provides further information to limit column or row access. <ul style="list-style-type: none">• In a column access policy the expression is the transformation of the column.• In a row access policy the expressions is the content of the WHERE clause. For example, the expression: <pre>=> ALTER ACCESS POLICY on customer_dimension for column customer_key length ('xxxxx') enable;</pre> limits access to strings in the customer_key column to a specific length.
ENABLE/DISABLE	Indicates whether to enable or disable the access policy at the table level.
COPY TO TABLE	Copies the existing access policy to the specified table. Do not use an expression when performing a COPY TO function.

Privileges

You must be a dbadmin user to alter an access policy.

Examples

Copy Access Policy from Table to Table

```
=> ALTER ACCESS POLICY on customer FOR COLUMN customer_number COPY TO TABLE customer_old;
```

Enable Access Policy

```
=> ALTER ACCESS POLICY on customer for rows where cid1>1 enable;
```

ALTER AUTHENTICATION

Modifies the settings for a specified authentication method.

Syntax

```
ALTER AUTHENTICATION auth_method_name {  
  | { ENABLE | DISABLE }  
  | { LOCAL | HOST [ { TLS | NO TLS } ] host_ip_address }  
  | RENAME TO new_auth_method_name  
  | METHOD value  
  | SET param=value [, ...]  
  | PRIORITY value
```

Parameters

Parameter Name	Description
<i>auth_method_name</i>	Name of the authentication method that you want to create. Type: VARCHAR
ENABLE DISABLE	Enable or disable the specified authentication method. Default: Enabled NOTE: When you perform an upgrade and use Kerberos authentication, you must manually set the authentication to ENABLE as it is disabled by default.

Parameter Name	Description
LOCAL HOST [{ TLS NO TLS } <i>host_ip_address</i>	<p>Specify that the authentication method applies to local or remote (HOST) connections.</p> <p>For authentication methods that use LDAP, specify whether or not LDAP uses Transport Layer Security (TLS).</p> <p>For remote (HOST) connections, you must specify the IP address of the host from which the user or application is connecting, VARCHAR.</p> <p>Vertica supports IPv4 and IPv6 addresses. For more information see IPv4 and IPv6 for Client Authentication.</p>
RENAME TO <i>new_auth_method_name</i>	<p>Rename the authentication record.</p> <p>Type: VARCHAR</p>
METHOD <i>value</i>	<p>The authentication method you are altering.</p>
SET <i>param=value</i>	<p>Set a parameter name and value for the authentication method that you are creating. Required only for LDAP and Ident authentication.</p> <p>ALTER AUTHENTICATION validates the parameters you enter.</p>
PRIORITY <i>value</i>	<p>If the user is associated with multiple authentication methods, the priority value specifies which authentication method Vertica tries first.</p> <p>Default:0</p> <p>Type: INTEGER</p> <p>Higher values indicate higher priorities. For example, a priority of 10 is higher than a priority of 5; priority 0 is the lowest possible value.</p> <p>For details, see Priorities for Client Authentication Methods.</p>

Privileges

Must have DBADMIN privileges.

Examples

Enabling and Disabling Authentication Methods

This example uses ALTER AUTHENTICATION to disable the v_ldap authentication method and then enable it again:

```
=> ALTER AUTHENTICATION v_ldap DISABLE;  
=> ALTER AUTHENTICATION v_ldap ENABLE;
```

Renaming Authentication Methods

This example renames the v_kerberos authentication method to K5 and enables it. All users who have been granted (associated with) the v_kerberos authentication method now have the K5 method granted instead.

```
=> ALTER AUTHENTICATION v_kerberos RENAME TO K5 ENABLE;
```

Modifying Authentication Parameters

This example sets the system user for ident1 authentication to user1:

```
=> CREATE AUTHENTICATION ident1 METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION ident1 SET system_users='user1';
```

When you set or modify LDAP or Ident parameters using ALTER AUTHENTICATION, Vertica validates them.

This example changes the IP address and specifies the parameters for an LDAP authentication method named Ldap1. Specify the bind parameters for the LDAP server. Vertica connects to the LDAP server, which authenticates the database client. If authentication succeeds, Vertica authenticates any users who have been associated with (granted) the Ldap1 authentication method on the designated LDAP server:

```
=> CREATE AUTHENTICATION Ldap1 METHOD 'ldap' HOST '172.16.65.196';  
=> ALTER AUTHENTICATION Ldap1 SET host='ldap://172.16.65.177',  
binddn_prefix='cn=', binddn_suffix=',dc=qa_domain,dc=com';
```

The next example specifies the parameters for an LDAP authentication method named Ldap2. Specify the LDAP search and bind parameters. Sometimes, Vertica does not have enough information to create the distinguished name (DN) for a user attempting to authenticate. In such cases, you must specify to use LDAP search and bind:

```
=> CREATE AUTHENTICATION Ldap2 METHOD 'ldap' HOST '172.16.65.196';
```

```
=> ALTER AUTHENTICATION Ldap2 SET basedn='dc=qa_domain,dc=com',  
    binddn='cn=Manager,dc=qa_domain,  
    dc=com',search_attribute='cn',bind_password='secret';
```

Changing the Authentication Method

This example changes the localpwd authentication from hash to trust:

```
=> CREATE AUTHENTICATION localpwd METHOD 'hash' LOCAL;  
=> ALTER AUTHENTICATION localpwd METHOD 'trust';
```

See Also

- [CREATE AUTHENTICATION](#)
- [DROP AUTHENTICATION](#)
- [GRANT \(Authentication\)](#)
- [REVOKE \(Authentication\)](#)
- [IPv4 and IPv6 for Client Authentication](#)

ALTER DATABASE

Use ALTER DATABASE to perform the following tasks:

- [Set](#) and [clear](#) database [configuration parameters](#).
- Drop all [fault groups](#) and their child fault groups from a database.
- Specify the subnet name of a public network to use for [import/export](#).
- Restore down nodes, and [revert active standby](#) nodes to standby status.

Syntax

```
ALTER DATABASE db-spec {  
... DROP ALL FAULT GROUP  
... | EXPORT ON { subnet-name | DEFAULT }  
... | RESET STANDBY  
... | SET [PARAMETER] parameter=value [,...]
```

```
... | CLEAR [PARAMETER] parameter[,...]  
}
```

Parameters

<i>db-spec</i>	Specifies the database to alter, one of the following: <ul style="list-style-type: none"> The database name DEFAULT: The current database
DROP ALL FAULT GROUP	Drops all fault groups defined on the specified database. The syntax for DROP ALL FAULT GROUP is singular for GROUP.
EXPORT ON	Specifies the network to use for importing and exporting data, one of the following: <ul style="list-style-type: none"> <i>subnet-name</i>: A subnet of the public network. DEFAULT: Specifies to use a private network.
RESET STANDBY	Restores any replaced nodes and reverts all active standby nodes to standby status. If any replaced nodes cannot resume activity, Vertica leaves the standby nodes in place.
SET [PARAMETER] <i>parameter=value</i>	Sets one or more configuration parameters to the specified value at the database level.
CLEAR [PARAMETER] <i>parameter</i>	Clears one or more specified configuration parameters at the database level.

Privileges

Superuser

Examples

Drop and restore all default groups:

```
=> ALTER DATABASE exampledb DROP ALL FAULT GROUP;  
ALTER DATABASE
```

Restore down nodes and revert all active standby nodes:

```
=> ALTER DATABASE exampledb RESET STANDBY;  
ALTER DATABASE
```

Set multiple configuration parameters:

```
=> ALTER DATABASE exampledb SET  
    AnalyzeRowCountInterval = 3600,  
    DBDCorrelationSampleRowCount = 3000,  
    ActivePartitionCount = 2;
```

Clear configuration parameters:

```
=> ALTER DATABASE exampledb CLEAR  
    AnalyzeRowCountInterval, DBDCorrelationSampleRowCount, ActivePartitionCount;
```

ALTER FAULT GROUP

Modifies an existing fault group. For example, use the ALTER FAULT GROUP statement to:

- Add a node to or drop a node from an existing fault group
- Add a child fault group to or drop a child fault group from a parent fault group
- Rename a fault group

Syntax

```
ALTER FAULT GROUP fault-group-name  
... [ ADD NODE node-name ]  
... [ DROP NODE node-name ]  
... [ ADD FAULT GROUP child-fault-group-name ]  
... [ DROP FAULT GROUP child-fault-group-name ]  
... [ RENAME TO new-fault-group-name ]
```

Parameters

<i>fault-group-name</i>	The existing fault group name you want to modify. Tip: For a list of all fault groups defined in the cluster, query the V_CATALOG.FAULT_GROUPS system table.
<i>node-name</i>	The node name you want to add to or drop from the existing (parent) fault group.
<i>child-fault-group-name</i>	The name of the child fault group you want to add to or remove from an existing parent fault group.
<i>new-fault-group-name</i>	The new name for the fault group you want to rename.

Privileges

Must be a superuser to alter a fault group.

Example

This example renames the parent0 fault group to parent100:

```
=> ALTER FAULT GROUP parent0 RENAME TO parent100;  
ALTER FAULT GROUP
```

You can verify the change by querying the [V_CATALOG.FAULT_GROUPS](#) system table:

```
=> SELECT member_name FROM fault_groups;  
  member_name  
-----  
v_exampledb_node0003  
parent100  
mygroup  
(3 rows)
```

See Also

- [CREATE FAULT GROUP](#)
- [V_CATALOG.FAULT_GROUPS](#)
- [V_CATALOG.CLUSTER_LAYOUT](#)
- [Fault Groups](#)
- [High Availability With Fault Groups](#)

ALTER FUNCTION (SQL)

Alters a user-defined SQL function.

Syntax

```
ALTER FUNCTION... [[db-name.]schema.]function-name([arg-type1], ...)  
... | SET SCHEMA new_schema  
... | RENAME TO new_name  
... | OWNER TO new_owner
```

Parameters

<code>[<i>db-name.</i>]schema</code>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<code>function-name</code>	<p>The name of the SQL function to alter. Each function requires an accompanying argument, so you must specify the argument type for each function.</p>
<code>RENAME TO <i>new_name</i></code>	<p>Specifies the new name of the function</p>
<code>SET SCHEMA <i>new_schema</i></code>	<p>Specifies the new schema name where the function resides.</p>
<code>OWNER TO <i>new_owner</i></code>	<p>Specifies the new owner of the function.</p>

Privileges

- Only a superuser or owner can alter a function.
- To rename a function (`ALTER FUNCTION RENAME TO`) the user must have `USAGE` and `CREATE` privilege on schema that contains the function.
- To specify a new schema (`ALTER FUNCTION SET SCHEMA`), the user must have `USAGE` privilege on schema that currently contains the function (old schema) and `CREATE` privilege on the schema to which the function will be moved (new schema).

Examples

This example renames a function called `SQL_one` to `SQL_two`:

```
=> ALTER FUNCTION SQL_one (int,int) RENAME TO SQL_two;
```

This example moves the `SQL_two` function to a new schema called `macros`:

```
=> ALTER FUNCTION SQL_two (int) SET SCHEMA macros;
```

This example assigns a new owner to `SQL_two`:

```
=> ALTER FUNCTION SQL_two (int, int) OWNER TO user1;
```

See Also

- [CREATE FUNCTION \(SQL Functions\)](#)
- [DROP FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Using User-Defined SQL Functions](#)

ALTER FUNCTION (UDF or UDT)

Alters a user defined function (UDF) or user-defined transform (UDT).

Syntax

```
ALTER FUNCTION... [[db-name.]schema.]function-name([arg-type1,] ...)  
... | SET SCHEMA new_schema  
... | RENAME TO new_name  
... | SET FENCED bool_val  
... | OWNER TO new_owner
```

Parameters

<code>[<i>db-name</i>.]<i>schema</i></code>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre> <p>If you do not specify a schema, the table is created in the default schema.</p>
<code><i>function-name</i></code>	The name of the UDF to alter. You must specify the argument type list, which may be empty.
<code>RENAME TO <i>new_name</i></code>	Specifies the new name of the function
<code>SET SCHEMA <i>new_schema</i></code>	Specifies the new schema name where the function resides.
<code>OWNER TO <i>new_owner</i></code>	Specifies the new owner of the function.
<code>SET FENCED <i>bool_val</i></code>	<p>A boolean value that specifies if Fenced Mode is enabled for this function. Valid values are:</p> <ul style="list-style-type: none">true - enables Fenced Modefalse - disables Fenced Mode <p>Fenced Mode is not available for User Defined Aggregates or User Defined Load.</p>

Privileges

- Only a superuser or owner can alter a function.
- To rename a function (ALTER FUNCTION RENAME TO) the user must have USAGE and CREATE privilege on schema that contains the function.
- To specify a new schema (ALTER FUNCTION SET SCHEMA), the user must have USAGE privilege on the schema that currently contains the function (old schema) and CREATE privilege on the schema to which the function will be moved (new schema).

Examples

This example renames a function called UDF_one to UDF_two:

```
=> ALTER FUNCTION UDF_one (int,int) RENAME TO UDF_two;
```

This example moves the UDF_two function to a new schema called macros:

```
=> ALTER FUNCTION UDF_two (int) SET SCHEMA macros;
```

This example disables Fenced Mode for the UDF_two function:

```
=> ALTER FUNCTION UDF_two (int, int) SET FENCED false;
```

See Also

- [CREATE FUNCTION \(UDF\)](#)
- [DROP FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)

ALTER LIBRARY

Replaces the library file (C++ .so file, Java JAR file, or R source file) associated with a UDX library defined in the Vertica catalog. Vertica automatically distributes the new file throughout the cluster. See [Developing User-Defined Extensions \(UDxs\)](#) in Extending Vertica for details. The UDXs defined in the catalog that reference the library automatically begin using the new library file.

Syntax

```
ALTER LIBRARY [schema.]library-name [DEPENDS 'support-path'] AS 'library-path';
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>library-name</i>	The library to alter. You can only alter an existing library.
[DEPENDS ' <i>support-path</i> ']	Indicates that the UDX library depends on one or more support libraries. Valid values: One or more absolute paths to the support libraries files, located in the initiator node's filesystem. Separate multiple paths with colons (:). Specify a directory containing multiple libraries using an asterisk wildcard (*). For example: '/home/mydir/mylibs/*'.
<i>library-path</i>	The absolute path in the initiator node's filesystem to the replacement library file.

Privileges

Superuser

Usage Considerations

- After the updated library file is distributed to all of the nodes in the Vertica cluster, every UDF that reference that library begin calling the code.
- Any nodes that are down or that are added to the cluster later automatically receive a copy of the updated library file when they join the cluster.
- The new library must be developed in the same programming language as the library file being replaced. For example, you cannot use this statement to replace a C++ library file with an R library file.
- Vertica does not compare the functions defined in the new library to ensure they match any currently-defined functions in the catalog. If you change the signature of a function in the library (for example, if you change the number and data types accepted by a UDSF defined in the library), calls to that function will likely generate errors. If your new library file changes the definition of a function, you must remove the function using [DROP FUNCTION](#) before using ALTER LIBRARY to load the new library. You can then recreate the function using its new signature.

Examples

This example shows how to update an already-defined library myFunctions with a new file.

```
=> ALTER LIBRARY myFunctions AS '/home/dbadmin/my_new_functions.so';
```

ALTER MODEL

Allows users to rename an existing model, change owner parameters, and set schema to the model.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
ALTER MODEL [[db-name.]schema.]model-name  
... OWNER TO owner-name  
... RENAME TO new-model-name  
... SET SCHEMA schema-name
```

Parameters

<i>db-name.schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema</pre>
<i>model-name</i>	The model to alter.
OWNER TO <i>owner-name</i>	Changes the model owner.
RENAME TO	Renames the model.
SET SCHEMA	Moves the model from one schema to another.

Privileges

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

Examples

This example shows how you can alter an existing model to rename the model.

```
=> ALTER MODEL mymodel RENAME to mykmeansmodel;  
ALTER MODEL
```

This example shows how you can alter an existing model to change the owner.

```
=> ALTER MODEL mykmeansmodel OWNER TO user1;  
ALTER MODEL
```

This example shows how you can alter an existing model to rename the model.

```
=> ALTER MODEL mykmeansmodel SET SCHEMA public;  
ALTER MODEL
```

See Also

- [Altering Models](#)

ALTER NETWORK INTERFACE

Lets you rename a network interface.

Syntax

```
ALTER NETWORK INTERFACE network-interface-name RENAME TO new-network-interface-name
```

The parameters are defined as follows:

<i>network-interface-name</i>	The name of the existing network interface.
<i>new-network-interface-name</i>	The new name for the network interface.

Privileges

Must be a superuser to alter a network interface.

Examples

This example shows how to rename a network interface.

```
=> ALTER NETWORK INTERFACE myNetwork RENAME TO myNewNetwork;
```

ALTER NODE

Sets and clears node-level configuration parameters on the specified node. ALTER NODE also performs the following management tasks:

- Changes the node type.
- Specifies the network interface of the public network on individual nodes that are used for import and export.
- Replaces a down node.

Syntax

```
ALTER NODE node-name {  
... EXPORT ON { network-interface | DEFAULT }  
... | [IS] node-type  
... | REPLACE [ WITH standby-node ]  
... | RESET  
... | SET [PARAMETER] parameter=value[,...]  
... | CLEAR [PARAMETER] parameter[,...]  
}
```

Parameters

<i>node-name</i>	The name of the node to alter.
[IS] <i>node-type</i>	Changes the node type, where <i>node-type</i> is one of the following: <ul style="list-style-type: none">• PERMANENT (default): A node that is used to store data.• EPHEMERAL: A node that is in transition from one type to another—typically, from PERMANENT to either STANDBY or EXECUTE.• STANDBY: A node that is reserved to replace any node when it goes down. When used as a replacement node, Vertica changes its type to PERMANENT. A standby node stores no segments or data until it is called to replace a down node. At

	<p>that time, Vertica changes its type to PERMANENT. For more information, see Active Standby Nodes.</p> <ul style="list-style-type: none"> EXECUTE: A node that is reserved for computation purposes only. An execute node contains no segments or data.
EXPORT ON	<p>Specifies the network to use for importing and exporting data, one of the following:</p> <ul style="list-style-type: none"> <i>network-interface</i>: The name of a network interface of the public network. DEFAULT: Use the default network interface of the public network, as specified by ALTER DATABASE.
REPLACE [WITH <i>standby-node</i>]	<p>Replaces the specified node with an available active standby node. If you omit the WITH clause, Vertica tries to find a replacement node from the same fault group as the down node.</p> <p>If you specify a node that is not down, Vertica ignores this statement.</p>
RESET	<p>Restores the specified down node and returns its replacement to standby status. If the down node is unable to resume activity, Vertica ignores this statement and leaves the standby node in place.</p>
SET [PARAMETER] <i>parameter=value</i>	<p>Sets one or more configuration parameters to the specified value at the node level.</p>
CLEAR [PARAMETER] <i>parameter</i>	<p>Clears one or more specified configuration parameters.</p>

Privileges

Superuser

Examples

Specify to use the default network interface of public network on `v_vmart_node0001` for import/export operations:

```
=> ALTER NODE v_vmart_node0001 EXPORT ON DEFAULT;
```

Replace down node `v_vmart_node0001` with an active standby node, then restore it:

```
=> ALTER NODE v_vmart_node0001 REPLACE WITH standby1;  
...  
=> ALTER NODE v_vmart_node0001 RESET;
```

Set and clear configuration parameter `MaxClientSessions`:

```
=> ALTER NODE v_vmart_node0001 SET MaxClientSessions = 0;  
...  
=> ALTER NODE v_vmart_node0001 CLEAR MaxClientSessions;
```

Set the node type as `EPHEMERAL`:

```
=> ALTER NODE v_vmart_node0001 IS EPHEMERAL;
```

ALTER NOTIFIER

Updates an existing notifier.

Note: To change the action URL associated with an existing identifier, [drop the notifier](#) and re-create it.

Syntax

```
ALTER NOTIFIER notifier-name parameter[...]
```

parameter

```
[NO] CHECK COMMITTED  
ENABLE | DISABLE  
IDENTIFIED BY uuid  
MAXMEMORYSIZE max-memory-size  
MAXPAYLOAD max-payLoad-size  
PARAMETERS 'adapter-params'
```

Parameters

<i>notifier-name</i>	Specifies the notifier to update.
[NO] CHECK COMMITTED	Specifies to wait for delivery confirmation before sending the next message in the queue. Not all messaging systems support delivery confirmation.
ENABLE DISABLE	Specifies whether to enable or disable the notifier.
IDENTIFIED BY <i>uuid</i>	Specifies the notifier's unique identifier. If set, all the messages published by this notifier have this attribute.
MAXMEMORYSIZE	<p>The maximum size of the internal notifier, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows:</p> <p><code>MAXMEMORYSIZE integer{K M G T}</code></p> <p>If the queue exceeds this size, the notifier drops excess messages.</p>
MAXPAYLOAD	<p>The maximum size of the message, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows:</p> <p><code>MAXPAYLOAD integer{K M G T}</code></p> <p>The default setting is adapter-specific—for example, 1 M for Kafka.</p>
PARAMETERS ' <i>adapter-params</i> '	<p>Specifies one or more optional adapter parameters that are passed as a string to the adapter. Adapter parameters apply only to the adapter associated with the notifier.</p> <p>For Kafka notifiers, refer to Configuring Kafka for Vertica.</p>

Privileges

Database Administrator

Examples

Update the settings on an existing notifier:

```
=> ALTER NOTIFIER my_dc_notifier
    ENABLE
    MAXMEMORYSIZE '2G'
    IDENTIFIED BY 'f8b0278a-3282-4e1a-9c86-e0f3f042a971'
    CHECK COMMITTED;
```

See Also

- [CREATE NOTIFIER](#)
- [DROP NOTIFIER](#)
- [Monitoring Vertica Using Notifiers](#) in the Administrator's Guide.

ALTER PROJECTION RENAME

Initiates a rename operation on the specified projection.

Syntax

```
ALTER PROJECTION [schema.]projection-name RENAME TO new-projection-name
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>projection-name</i>	The projection to change. See Projection Naming for

	projection name conventions.
<i>new-projection-name</i>	The new projection name.

Privileges

The following privileges are required:

- Ownership of the projection's anchor table
- USAGE and CREATE privileges on the projection's schema

Examples

This example shows how to rename a projection.

```
=> CREATE TABLE tbl (x integer, y integer);  
CREATE TABLE  
  
=> CREATE PROJECTION tbl_p AS SELECT x from tbl;  
CREATE PROJECTION  
  
=> ALTER PROJECTION tbl_p RENAME to new_tbl_p;  
ALTER PROJECTION
```

See Also

[CREATE PROJECTION](#)

ALTER PROFILE

Changes a [profile](#). Only a database superuser can alter a profile.

Syntax

```
ALTER PROFILE name LIMIT  
... [PASSWORD_LIFE_TIME {life-limit | DEFAULT | UNLIMITED}]  
... [PASSWORD_GRACE_TIME {grace-period | DEFAULT | UNLIMITED}]  
... [FAILED_LOGIN_ATTEMPTS {login-limit | DEFAULT | UNLIMITED}]  
... [PASSWORD_LOCK_TIME {lock-period | DEFAULT | UNLIMITED}]
```

```
... [PASSWORD_REUSE_MAX {reuse-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_TIME {reuse-period | DEFAULT | UNLIMITED}]
... [PASSWORD_MAX_LENGTH {max-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LENGTH {min-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LETTERS {min-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_UPPERCASE_LETTERS {min-cap-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LOWERCASE_LETTERS {min-lower-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_DIGITS {min-digits | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_SYMBOLS {min-symbols | DEFAULT | UNLIMITED}]
```

Note: For all parameters, the special value DEFAULT means the parameter is inherited from the DEFAULT profile.

Parameters

Name	Description	Meaning of UNLIMITED value
<i>name</i>	The name of the profile to create, where <i>name</i> conforms to conventions described in Identifiers .	N/A
PASSWORD_LIFE_TIME <i>life-limit</i>	Integer number of days a password remains valid. After the time elapses, the user must change the password (or will be warned that their password has expired if PASSWORD_GRACE_TIME is set to a value other than zero or UNLIMITED).	Passwords never expire.
PASSWORD_GRACE_TIME <i>grace-period</i>	Integer number of days the users are allowed to login (while being issued a warning message) after their passwords are older than the PASSWORD_LIFE_TIME. After this period expires, users are forced to change their passwords on login if they have not done so after their password expired.	No grace period (the same as zero)
FAILED_LOGIN_ATTEMPTS <i>login-limit</i>	Number of consecutive failed login	Accounts are

Name	Description	Meaning of UNLIMITED value
	attempts permitted before locking a user's account.	never locked, no matter how many failed login attempts are made.
PASSWORD_LOCK_TIME <i>lock-period</i>	Integer value setting the number of days an account is locked after a user's account is locked after too many failed login attempts. After the PASSWORD_LOCK_TIME has expired, the account is automatically unlocked.	Accounts locked because of too many failed login attempts are never automatically unlocked. They must be manually unlocked by the database superuser.
PASSWORD_REUSE_MAX <i>reuse-limit</i>	The number of password changes that need to occur before the current password can be reused.	Users are not required to change passwords a certain number of times before reusing an old password.
PASSWORD_REUSE_TIME <i>reuse-period</i>	The integer number of days that must pass after a password has been set before it can be reused.	Password reuse is not limited by time.
PASSWORD_MAX_LENGTH <i>max-length</i>	The maximum number of characters allowed in a password. Value must be in the range of 8 to 100.	Passwords are limited to 100 characters.
PASSWORD_MIN_LENGTH <i>min-length</i>	The minimum number of characters required in a password. Valid range is 0 to <i>max-length</i> .	Equal to <i>max-length</i> .
PASSWORD_MIN_LETTERS <i>min-of-letters</i>	Minimum number of letters (a-z	0 (no minimum).

Name	Description	Meaning of UNLIMITED value
	and A-Z) that must be in a password. Valid ranged is 0 to <i>max-Length</i> .	
PASSWORD_MIN_UPPERCASE_LETTERS <i>min-cap-Letters</i>	Minimum number of capital letters (A-Z) that must be in a password. Valid range is is 0 to <i>max-Length</i> .	0 (no minimum).
PASSWORD_MIN_LOWERCASE_LETTERS <i>min-Lower-Letters</i>	Minimum number of lowercase letters (a-z) that must be in a password. Valid range is is 0 to <i>max-Length</i> .	0 (no minimum).
PASSWORD_MIN_DIGITS <i>min-digits</i>	Minimum number of digits (0-9) that must be in a password. Valid range is is 0 to <i>max-Length</i> .	0 (no minimum).
PASSWORD_MIN_SYMBOLS <i>min-symbols</i>	Minimum number of symbols (any printable non-letter and non-digit character, such as \$, #, @, and so on) that must be in a password. Valid range is is 0 to <i>max-Length</i> .	0 (no minimum).

Privileges

Must be a superuser to alter a profile.

Note: Only the profile settings for how many failed login attempts trigger [Account Locking](#) and how long accounts are locked have an effect on password authentication methods such as LDAP or GSS. All password [complexity](#), reuse, and [lifetime settings](#) affect only passwords that Vertica manages.

Example

```
ALTER PROFILE sample_profile LIMIT FAILED_LOGIN_ATTEMPTS 3;
```

See Also

- [CREATE PROFILE](#)
- [DROP PROFILE](#)
- [Creating a Database Name and Password](#)

ALTER PROFILE RENAME

Rename an existing profile.

Syntax

```
ALTER PROFILE name RENAME TO newname;
```

Parameters

<i>name</i>	The current name of the profile.
<i>newname</i>	The new name for the profile.

Privileges

Must be a superuser to alter a profile.

Examples

This example shows how to rename an existing profile.

```
ALTER PROFILE sample_profile RENAME TO new_sample_profile;
```

See Also

- [ALTER PROFILE](#)
- [CREATE PROFILE](#)
- [DROP PROFILE](#)

ALTER RESOURCE POOL

Modifies an existing resource pool by setting one or more parameters.

Syntax

```
ALTER RESOURCE POOL pool-name [ parameter-name setting ]...
```

Parameters

Note: You can set all resource pool parameters to their DEFAULT value. The [V_CATALOG.RESOURCE_POOL_DEFAULTS](#) system table contains default parameter values. Query this table to determine default settings for all resource pools.

Default values specified in this table pertain only to user-defined resource pools. For built-in pool default values, see [Built-In Pool Configuration](#).

<i>pool-name</i>	The name of the resource pool. Built-in pool names cannot be used for user-defined pools.
<i>parameter-name</i>	The parameter to set, listed below.
CASCADE TO	Specifies a secondary resource pool for executing queries that exceed the RUNTIMECAP setting of their assigned resource pool: CASCADE TO <i>secondary-pool</i>
CPUAFFINITYMODE	Specifies whether the resource pool has exclusive or shared use of the CPUs specified in CPUAFFINITYSET :

	<pre>CPUAFFINITYMODE { SHARED EXCLUSIVE ANY }</pre> <ul style="list-style-type: none"> • SHARED: Queries that run in this pool share its CPUAFFINITYSET CPUs with other Vertica resource pools. • EXCLUSIVE: Dedicates CPUAFFINITYSET CPUs to this resource pool only, and excludes other Vertica resource pools. If CPUAFFINITYSET is set as a percentage, then that percentage of CPU resources available to Vertica is assigned solely for this resource pool. • ANY (default): Queries in this resource pool can run on any CPU, invalid if CPUAFFINITYSET designates CPU resources.
<p>CPUAFFINITYSET</p>	<p>Specifies which CPUs are available to this resource pool. All cluster nodes must have the same number of CPUs. The CPU resources assigned to this set are unavailable to general resource pools.</p> <pre>CPUAFFINITYSET { 'cpu-index[,...]' 'cpu-index_i-cpu-index_n' 'integer%' NONE }</pre> <ul style="list-style-type: none"> • <i>cpu-index</i>[, ...]: Dedicates one or more comma-delimited CPUs to this pool. • <i>cpu-index_i-cpu-index_n</i>: Dedicates a range of contiguous CPU indexes to this pool • <i>integer%</i>: Percentage of all available CPUs to use for this pool. Vertica rounds this percentage down to include whole CPU units. • NONE (default): No affinity set is assigned to this resource pool. The queries associated with this pool are executed on any CPU.

<p>EXECUTIONPARALLELISM</p>	<p>Limits the number of threads used to process any single query issued in this resource pool.</p> <pre>EXECUTIONPARALLELISM { <i>integer</i> AUTO }</pre> <ul style="list-style-type: none"> • <i>integer</i>: A value between 1 and the number of cores. Setting this parameter to a reduced value increases throughput of short queries issued in the pool, especially if the queries are executed concurrently. • AUTO (default): Vertica sets this value based on the number of cores, available memory, and amount of data in the system. Unless memory is limited, or the amount of data is very small, Vertica sets this value to the number of cores on the node.
<p>MAXCONCURRENCY</p>	<p>Sets the maximum number of concurrent execution slots available to the resource pool, across the cluster:</p> <pre>MAXCONCURRENCY { <i>integer</i> NONE }</pre> <p>NONE (default) specifies unlimited number of concurrent execution slots.</p>
<p>MAXMEMORYSIZE</p>	<p>The maximum size per node the resource pool can grow by borrowing memory from the GENERAL pool:</p> <pre>MAXMEMORYSIZE { '<i>integer</i>%' '<i>integer</i>{K M G T}' NONE }</pre> <ul style="list-style-type: none"> • <i>integer</i>?: Percentage of total memory • <i>integer</i>{K M G T}: Amount of memory in kilobytes, megabytes, gigabytes, or terabytes • NONE: Unlimited; pool can borrow any amount of available memory from the GENERAL pool.
<p>MEMORYSIZE</p>	<p>The amount of total memory available to the Vertica resource manager that is allocated to this pool per node:</p> <pre>MEMORYSIZE { '<i>integer</i>%' '<i>integer</i>{K M G T}' }</pre>

	<ul style="list-style-type: none"> • <i>integer</i>?: Percentage of total memory • <i>integer</i>{K M G T}: Amount of memory in kilobytes, megabytes, gigabytes, or terabytes <p>Default: 0%. No memory allocated, the resource pool borrows memory from the GENERAL pool.</p>
<p>PLANNEDCONCURRENCY</p>	<p>Specifies the preferred number queries to execute concurrently in the resource pool. This setting applies to the entire cluster:</p> <pre>PLANNEDCONCURRENCY { <i>integer</i> AUTO }</pre> <ul style="list-style-type: none"> • <i>integer</i>: The preferred number of concurrently executing queries. When possible, query resource budgets are limited to allow this level of concurrent execution. • AUTO (default): Value is calculated automatically at query runtime. Vertica sets this parameter to the lower of these two calculations, but never less than 4: <ul style="list-style-type: none"> ■ Number of logical cores ■ Memory divided by 2GB <p>For clusters where the number of logical cores differs on different nodes, AUTO can apply differently on each node. Distributed queries run like the minimal effective planned concurrency. Single node queries run with the planned concurrency of the initiator.</p> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <p>Tip: Change this parameter only after evaluating performance over a period of time.</p> </div>
<p>PRIORITY</p>	<p>Specifies priority of queries in this pool when they compete for resources in the GENERAL pool:</p> <pre>PRIORITY { <i>integer</i> HOLD }</pre> <ul style="list-style-type: none"> • <i>integer</i>: A negative or positive integer value, where higher numbers denote higher priority: <ul style="list-style-type: none"> ■ User-defined pools: -100 to 100

	<ul style="list-style-type: none"> ■ Built-in pools SYSQUERY, RECOVERY, and TM: -110 to 110 ● HOLD: Sets priority to -999. Queries in this pool are queued until QUEUE_TIMEOUT is reached. <p>Default: 0</p>
<p>QUEUE_TIMEOUT</p>	<p>Specifies how long a request can wait for pool resources before it is rejected:</p> <p><code>QUEUE_TIMEOUT { integer NONE }</code></p> <ul style="list-style-type: none"> ● <i>integer</i>: Maximum wait time in seconds ● NONE: No maximum wait time, request can be queued indefinitely. <p>Default: 300 seconds</p>
<p>RUNTIMECAP</p>	<p>Prevents runaway queries by setting the maximum time a query in the pool can execute. If a query exceeds this setting, it tries to cascade to a secondary pool:</p> <p><code>RUNTIMECAP { interval NONE }</code></p> <ul style="list-style-type: none"> ● <i>interval</i>: An interval of 1 minute or 100 seconds; should not exceed one year. ● NONE: No time limit on queries running in this pool. <p>To specify a value in days, provide an integer value. To provide a value less than one day, provide the interval in the format <code>hours:minutes:seconds</code>. For example a value of <code>1:30:00</code> would equal 90 minutes.</p> <p>If the user or session also has a RUNTIMECAP, the shorter limit applies.</p>
<p>RUNTIMEPRIORITY</p>	<p>Determines how the resource manager should prioritize dedication of run-time resources (CPU, I/O bandwidth) to queries already running in this resource pool:</p> <p><code>RUNTIMEPRIORITY { HIGH MEDIUM LOW }</code></p> <p>Default: MEDIUM</p>
<p>RUNTIMEPRIORITYTHRESHOLD</p>	<p>Specifies in seconds a time limit in which a query must</p>

	<p>finish before the resource manager assigns to it the resource pool's RUNTIMEPRIORITY. All queries begin running at a HIGH priority. When a query's duration exceeds this threshold, it is assigned the RUNTIMEPRIORITY of the resource pool.</p> <p><code>RUNTIMEPRIORITYTHRESHOLD</code> <i>seconds</i></p> <p>Default: 2</p>
--	--

Privileges

The following parameters require superuser privileges on the resource pool:

- CASCADE TO
- CPUAFFINITYSET
- CPUAFFINITYMODE
- MAXMEMORYSIZE
- PRIORITY
- QUEUETIMEOUT

The following parameters require UPDATE privileges on the resource pool:

- MAXCONCURRENCY
- PLANNEDCONCURRENCY
- SINGLEINITIATOR

Examples

This example shows how to alter resource pool `ceo_pool` by setting the priority to 5.

```
=> ALTER RESOURCE POOL ceo_pool PRIORITY 5;
```

This example shows how to designate a secondary pool for the `ceo_pool`.

```
=> CREATE RESOURCE POOL second_pool;  
=> ALTER RESOURCE POOL ceo_pool CASCADE TO second_pool;
```

See Also

- [CREATE RESOURCE POOL](#)
- [CREATE USER](#)
- [DROP RESOURCE POOL](#)
- [RESOURCE_POOL_STATUS](#)
- [SET SESSION RESOURCE_POOL](#)
- [SET SESSION MEMORYCAP](#)
- [Managing Workloads](#)

ALTER ROLE RENAME

Rename an existing role.

NOTE: You cannot use the ALTER ROLE RENAME command on a role added to the Vertica database with the LDAPLink service.

Syntax

```
ALTER ROLE name RENAME TO new_name;
```

Parameters

<i>name</i>	The current name of the role that you want to rename.
<i>new_name</i>	The new name for the role.

Privileges

Must be a superuser to rename a role.

Example

```
=> ALTER ROLE applicationadministrator RENAME TO appadmin;  
ALTER ROLE
```

See Also

- [CREATE ROLE](#)
- [DROP ROLE](#)

ALTER SCHEMA

Renames one or more existing schemas.

Caution: Renaming a schema referenced by a view causes the view to fail unless another schema is created to replace it.

Syntax

```
ALTER SCHEMA [schema-name[,...]] RENAME TO new-schema-name [...]  
[ DEFAULT {INCLUDE | EXCLUDE} SCHEMA PRIVILEGES ]
```

Parameters

<i>schema-name</i>	Specifies a schema to rename.
<i>new-schema-name</i>	Specifies one or more new schema names. When renaming schemas, be

	<p>sure that:</p> <ul style="list-style-type: none"> • New schema names are not already in use. • The number of schemas to rename corresponds with the number of new schema names. <p>Vertica applies the new schema names in the RENAME TO parameter atomically, renaming either all schemas or none. For example, if the number of schemas to rename does not match the number of new names you supply, no schemas are renamed.</p>
<p>DEFAULT {INCLUDE EXCLUDE} SCHEMA PRIVILEGES</p>	<p>Specifies whether to enable or disable default privileges for new tables in the renamed schema.</p> <p>INCLUDE SCHEMA PRIVILEGES specifies to grant tables in the renamed schema the same privileges granted to that schema. This option has no effect on tables that were created in the schema before it was renamed.</p> <p>For more information see Grant Inherited Privileges.</p>

Privileges

Schema owner or user requires CREATE privilege on the database

Swapping Schemas

Renaming schemas is useful for swapping schemas without actually moving data. To facilitate the swap, enter a non-existent, temporary placeholder schema. For example, the following ALTER SCHEMA statement uses the temporary schema *temps* to facilitate swapping schema S1

with schema *S2*. In this example, *S1* is renamed to *temps*. Then *S2* is renamed to *S1*. Finally, *temps* is renamed to *S2*.

```
ALTER SCHEMA S1, S2, temps RENAME TO temps, S1, S2;
```

Examples

The following example renames schemas *S1* and *S2* to *S3* and *S4*, respectively:

```
=> ALTER SCHEMA S1, S2 RENAME TO S3, S4;
```

This example sets the default behavior for new table *t2* to automatically inherit the schema's privileges:

```
=> ALTER SCHEMA s1 DEFAULT INCLUDE SCHEMA PRIVILEGES;  
=> CREATE TABLE s1.t2 (i, int);
```

This example sets the default for new tables to not automatically inherit privileges from the schema:

```
=> ALTER SCHEMA s1 DEFAULT EXCLUDE SCHEMA PRIVILEGES;
```

See Also

- [CREATE SCHEMA](#)
- [DROP SCHEMA](#)

ALTER SEQUENCE

Changes the attributes of an existing sequence. All changes take effect in the next database session. Any parameters not set by an `ALTER SEQUENCE` statement retain their previous settings.

Syntax

General Usage

```
ALTER SEQUENCE [schema.]sequence-name
... [ INCREMENT [ BY ] increment-value ]
... [ MINVALUE minvalue | NO MINVALUE ]
... [ MAXVALUE maxvalue | NO MAXVALUE ]
... [ RESTART [ WITH ] restart-value ]
... [ CACHE value | NO CACHE ]
... [ CYCLE | NO CYCLE ]
```

Changing Name, Schema, and Ownership

```
ALTER SEQUENCE [schema.]sequence-name {
... RENAME TO new-name
... | SET SCHEMA new-schema-name]
... | OWNER TO new-owner-name
}
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre> <p>If you do not specify a schema, the table is created in the default schema.</p>
<i>sequence-name</i>	The name of the sequence to alter. The name must be unique among sequences, tables, projections, and views.
RENAME TO <i>new-name</i>	Renames a sequence within the same schema. To move a named sequence, use SET SCHEMA.
OWNER TO <i>new-owner-name</i>	<p>Reassigns the current sequence owner to the specified owner.</p> <p>Only the sequence owner or a superuser can change ownership, and reassignment does not transfer grants from the original owner to the new owner (grants made by the original owner are dropped).</p>
SET SCHEMA <i>new-schema-name</i>	Moves a named sequence between schemas.

<p>INCREMENT [BY] <i>increment-value</i></p>	<p>Modifies how much to increment or decrement the current sequence to create a new value. A positive value increments an ascending sequence, and a negative value decrements the sequence.</p>
<p>MINVALUE <i>minvalue</i> NO MINVALUE</p>	<p>Modifies the minimum value a sequence can generate. If you change this value and the current value exceeds the range, the current value is changed to the minimum value if increment is greater than zero, or to the maximum value if increment is less than zero.</p>
<p>MAXVALUE <i>maxvalue</i> NO MAXVALUE</p>	<p>Modifies the maximum value for the sequence. If you change this value and the current value exceeds the range, the current value is changed to the minimum value if increment is greater than zero, or to the maximum value if increment is less than zero.</p>
<p>RESTART [WITH] <i>restart-value</i></p>	<p>Changes the current value of the sequence to <i>restart-value</i>. The next call to NEXTVAL returns <i>restart-value</i>.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Caution: Using ALTER SEQUENCE to set a sequence start value below its current value can result in duplicate keys.</p> </div>
<p>CACHE <i>value</i> NO CACHE</p>	<p>Modifies how many sequence numbers are preallocated and stored in memory for faster access. The default is 250,000 with a minimum value of 1. Specifying a value of 1 indicates that only one value can be generated at a time, since no cache is assigned. Alternatively, you can specify NO CACHE.</p>
<p>CYCLE NO CYCLE</p>	<p>Specifies whether the sequence can wrap when its minimum or maximum values are reached:</p> <ul style="list-style-type: none"> • CYCLE: Allows the sequence to wrap around when the maxvalue or minvalue is reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated is the minvalue or maxvalue, respectively. • NO CYCLE (default): Calls to NEXTVAL return an error after the sequence reaches its maximum/minimum value.

Privileges

- To rename a schema: sequence owner and USAGE and CREATE privileges on the schema.
- To move a named sequence between schemas: sequence owner and USAGE privilege on the schema that currently contains the sequence (old schema) and CREATE privilege on new schema to contain the sequence.

Examples

See [Altering Sequences](#) and [Changing Sequence Ownership](#) in the Administrator's Guide.

See Also

[CREATE SEQUENCE](#)

ALTER SESSION

Use ALTER SESSION to set and clear session-level configuration parameter values for the current session.

Syntax

```
ALTER SESSION {  
... SET [PARAMETER] parameter-name=value [,...]  
... | CLEAR [PARAMETER] parameter-name [,...]  
... | SET UDPARAMETER [ FOR { namespace | libname } ] key=value [,...]  
... | CLEAR UDPARAMETER [ FOR { namespace | libname } ] key [,...]  
}
```

Parameters

SET [PARAMETER]	Sets one or more configuration parameters to the specified value.
-----------------	---

CLEAR [PARAMETER]	Clears one or more specified configuration parameters .
SET UDPARAMETER	Sets a user-defined session parameter, can be used in combination with a UDx. If you do not specify a name space, Vertica uses the PUBLIC name space. Key size values: <ul style="list-style-type: none">• Set from client side: 128 characters• Set from UDx side: Unlimited
CLEAR UDPARAMETER	Clears one or more user-defined session parameters.

Privileges

None

Examples

Set and clear a parameter

- Force all UDxes that support fenced mode to run in fenced mode, even if their definition specified is NOT FENCED:

```
=> ALTER SESSION SET ForceUDxFencedMode = 1;
```

- Clear ForceUDxFencedMode at the session level. The value now reflects the default value (0).

```
=> ALTER SESSION CLEAR ForceUDxFencedMode;
```

Set and clear a user-defined parameter

- Set the value of RowCount, in the MyLibrary library and PUBLIC namespace, to 25.

```
=> ALTER SESSION SET UDPARAMETER FOR MyLibrary RowCount = 25;
```

- Clear RowCount at the session level. The value now reflects the default value (0).

```
=> ALTER SESSION CLEAR UDPARAMETER FOR MyLibrary RowCount;
```

See Also

[SESSION_PARAMETERS](#)

ALTER SUBNET

Renames an existing subnet.

Syntax

```
ALTER SUBNET subnet-name RENAME TO new-subnet-name
```

Parameters

The parameters are defined as follows:

<i>subnet-name</i>	The name of the existing subnet.
<i>new-subnet-name</i>	The new name for the subnet.

Privileges

Must be a superuser to alter a subnet.

Examples

This example renames a subnet.

```
=> ALTER SUBNET mysubnet RENAME TO myNewSubnet;
```

ALTER TABLE

Modifies the metadata of an existing table. The changes are auto-committed.

You cannot modify the metadata of an anchor table for a live aggregate projection or Top-K projection.

General Usage

```
ALTER TABLE [schema.]table-name {  
... ADD COLUMN column-name data-type  
    [ column-constraint ]  
    [ ENCODING encoding-type ]  
    [ RESTRICT | CASCADE ]  
    [ PROJECTIONS (projection-name [,...]) ]  
... | ADD table-constraint  
... | ALTER COLUMN column-name column-setting  
... | ALTER CONSTRAINT constraint-name { ENABLED | DISABLED }  
... | DROP CONSTRAINT constraint-name [ CASCADE | RESTRICT ]  
... | DROP [ COLUMN ] column-name [ CASCADE | RESTRICT ]  
... | FORCE OUTER integer  
... | OWNER TO owner-name  
... | PARTITION BY partition-clause [ REORGANIZE ]  
... | { INCLUDE | EXCLUDE | MATERIALIZE } [ SCHEMA ] PRIVILEGES  
... | REMOVE PARTITIONING  
... | RENAME [ COLUMN ] name TO new-name  
... | REORGANIZE  
... | SET DATA TYPE datatype  
... | SET SCHEMA schema-name  
... | SET STORAGE load-option  
}
```

Table Renaming

```
ALTER TABLE [schema.]table-name[,...] RENAME TO new-table-name[,...]
```

Parameters

schema

Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:

	myschema.thisDBObject
<i>table-name</i>	The table to alter.
ADD COLUMN	<p>Adds a column to the table and to its superprojections:</p> <pre>ADD COLUMN <i>column-name</i> <i>datatype</i> [<i>column-constraint</i>] [ENCODING <i>encoding-type</i>] [RESTRICT CASCADE] [PROJECTIONS (<i>projection-name</i> [,...])]</pre> <p>You can qualify the new column definition with one of these options:</p> <ul style="list-style-type: none"> • <i>column-constraint</i> specifies a column constraint as follows: <pre>[CONSTRAINT <i>constraint-name</i>] { {NULL NOT NULL} DEFAULT <i>expression</i> }</pre> • ENCODING specifies the column's encoding type, by default set to AUTO. • RESTRICT (default) adds the new column to pre-join projections only if they are anchored to the updated table. CASCADE updates all pre-join projections where the table is specified, regardless of whether they are anchored to it. • PROJECTIONS specifies adding the new column to one or more existing projections. <p>Note: When you add a new column to a table, use the PROJECTIONS option to simultaneously add the column to one or more existing projections (non-superprojections). See Adding Table Columns.</p>
ADD <i>table-constraint</i>	<p>Adds a constraint to a table that does not have any associated projections.</p> <p>See About Constraints in the Administrator's Guide.</p>
ALTER COLUMN	<p>Alters a setting for <i>column-name</i>:</p> <pre>ALTER COLUMN <i>column-name</i> <i>column-setting</i></pre> <p><i>column-setting</i> can be one of the following:</p>

	<ul style="list-style-type: none"> • SET DEFAULT <i>expression</i> sets column values to the specified expression. <div data-bbox="586 323 1401 688" style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p>Note: Altering an existing table column to specify a DEFAULT expression has no effect on existing values in that column. Vertica applies the DEFAULT expression only on new rows when they are added to the table, through load operations such as INSERT and COPY. To refresh the entire column with its DEFAULT expression, update the column as follows:</p> <pre>UPDATE <i>table-name</i> SET <i>column-name</i>=DEFAULT;</pre> </div> <ul style="list-style-type: none"> • DROP DEFAULT drops the DEFAULT setting. • SET USING <i>expression</i> specifies to set this column's values from the result set returned by <i>expression</i>. When you make this change, Vertica automatically calls the function REFRESH_COLUMNS, which populates the column from the result set returned by <i>expression</i>. Thereafter, the column contents are refreshed only on explicit calls to REFRESH_COLUMNS. • DROP SET USING drops the SET USING setting. • SET DEFAULT USING <i>expression</i> combines SET DEFAULT and SET USING clauses, setting both to the same expression. The DDL for the column specifies both clauses. • DROP DEFAULT USING drops the column's DEFAULT and SET USING settings. • SET NOT NULL disallows inserting null values in the column. • DROP NOT NULL allows null values in the column. • SET DATA TYPE <i>datatype</i> resets the column data type, if Vertica supports the change. No restrictions apply to changing data types of external tables.
ALTER CONSTRAINT	<p>Applies to automatic enforcement of primary key, unique key, and check constraints:</p> <div data-bbox="545 1808 1401 1871" style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <pre>ALTER CONSTRAINT <i>constraint-name</i> {ENABLED DISABLED}</pre> </div>

	See Altering Constraint Enforcement in the Administrator's Guide.
DROP CONSTRAINT	<p>Drops the specified table constraint from the table:</p> <pre>DROP CONSTRAINT <i>constraint-name</i> [CASCADE RESTRICT]</pre> <p>You can qualify DROP CONSTRAINT with one of these options:</p> <ul style="list-style-type: none"> • CASCADE : Drops a constraint and all dependencies in other tables. • RESTRICT: Does not drop a constraint if there are dependent objects. Same as the default behavior. <p>Dropping a table constraint has no effect on views that reference the table.</p>
DROP [COLUMN]	<p>Drops the specified column from the table and that column's ROS containers:</p> <pre>DROP [COLUMN] <i>column-name</i> [CASCADE RESTRICT]</pre> <p>You can qualify DROP COLUMN with one of these options:</p> <ul style="list-style-type: none"> • CASCADE is required if the table column to drop has dependencies. • RESTRICT drops the column only from the given table. <p>See Dropping Table Columns in the Administrator's Guide.</p>
FORCE OUTER <i>integer</i>	Specifies whether a table is joined to another as an inner or outer input. For details, see Controlling Join Inputs in Analyzing Data.
OWNER TO <i>owner-name</i>	Changes the table owner. See Changing Table Ownership in the Administrator's Guide.
PARTITION BY	<p>Specifies how to repartition table data storage, where partition-expression resolves to a value derived from one or more table columns:</p> <pre>PARTITION BY <i>partition-expression</i> [REORGANIZE]</pre> <p>This option is invalid for external tables.</p> <p>If a partitioning clause includes REORGANIZE, Vertica immediately</p>

	<p>implements the partition clause by initiating a mergeout operation. This operation drops existing partition keys and repartitions table data.</p> <p>Caution: If <code>PARTITION BY</code> omits <code>REORGANIZE</code>, table partitioning only applies to new data. This can put table partitioning in an inconsistent state and adversely affect performance. Vertica repartitions all table data only with a mergeout, which you can initiate with <code>ALTER TABLE . . . REORGANIZE</code>, or with functions DO_TM_TASK or PARTITION_TABLE.</p>
<pre>{ INCLUDE EXCLUDE MATERIALIZE } [SCHEMA] PRIVILEGES</pre>	<p>Specifies default inheritance of schema privileges for this table:</p> <ul style="list-style-type: none"> • <code>EXCLUDE [SCHEMA] PRIVILEGES</code> (default) disables inheritance of privileges from the schema. • <code>INCLUDE [SCHEMA] PRIVILEGES</code> grants the table the same privileges granted to its schema. • <code>MATERIALIZE</code>: Copies grants to the table and creates a <code>GRANT</code> object on the table. This disables the inherited privileges flag on the table, so you can: <ul style="list-style-type: none"> ■ Grant more specific privileges at the table level ■ Use schema-level privileges as a template ■ Move the table to a different schema ■ Change schema privileges without affecting the table <p>Note: If inherited privileges are disabled at the database level, schema privileges can still be materialized.</p> <p>For more information see Grant Inherited Privileges.</p>
<pre>REMOVE PARTITIONING</pre>	<p>Immediately removes partitioning on a table. The ROS containers are not immediately altered, but they are later cleaned by the Tuple Mover.</p>
<pre>RENAME [COLUMN]</pre>	<p>Renames the specified column within the table. See Renaming Columns in the Administrator's Guide.</p>

REORGANIZE	Initiates a mergeout operation, which repartitions table data according to the table's <code>PARTITION BY</code> clause. See also <code>PARTITION BY</code> , above.
SET DATA TYPE <i>datatype</i>	Changes the column's data type to any type whose conversion does not require storage reorganization. See Changing a Column Data Type in the Administrator's Guide.
SET SCHEMA	Moves the table from one schema to another. Vertica automatically moves all projections that are anchored to the source table to the destination schema. For details, see Moving Tables to Another Schema in the Administrator's Guide.
SET STORAGE <i>Load-method</i>	Specifies default load behavior for all DML operations on this table, such as <code>INSERT</code> and <code>COPY</code> , one of the following: <ul style="list-style-type: none"> • <code>AUTO</code> (default): Initially loads data into WOS, suitable for smaller bulk loads. • <code>DIRECT</code>: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads. • <code>TRICKLE</code>: Loads data only into WOS, suitable for frequent incremental loads. For details, see Choosing a Load Method in the Administrator's Guide.
RENAME TO	Renames one or more tables: <pre>RENAME TO <i>new-table-name</i>[, ...]</pre> The following requirements apply: <ul style="list-style-type: none"> • The new table name must be unique among all tables, views and projections within the same schema. • If you specify multiple tables to rename, the source and target lists must have the same number of names. • Renaming a table requires <code>USAGE</code> and <code>CREATE</code> privileges on the schema that contains the view.

Privileges

The following privileges are required:

- Table owner or superuser, and USAGE privileges on the table schema to make the following changes:
 - Add, drop, rename, or alter column.
 - Add or drop a constraint.
 - Partition or repartition the table.

Renaming a table requires USAGE and CREATE privilege on the table schema.

Moving a table to a new schema requires:

- USAGE privilege on the old schema
- CREATE privilege on new schema

Locked Tables

If the operation cannot obtain an [O lock](#) on the target table, Vertica tries to close any internal [tuple mover](#) (TM) sessions that are running on that table. If successful, the operation can proceed. Explicit TM operations that are running in user sessions do not close. If an explicit TM operation is running on the table, the operation proceeds only when the TM operation is complete.

See Also

- [Managing Tables](#)
- [Altering Table Definitions](#)
- [Adding Table Columns](#)

Table-Constraint

Adds a constraint to table metadata. You can specify table constraints with [CREATE TABLE](#), or add a constraint to an existing table with [ALTER TABLE](#). For details, see [Adding Constraints](#) in the Administrator's Guide.

Note: Adding a constraint to a table that is referenced in a view does not affect the view.

Syntax

```
[ CONSTRAINT constraint-name ]
{
... PRIMARY KEY (column[,... ]) [ ENABLED | DISABLED ]
... | FOREIGN KEY (column[,... ]) REFERENCES table [(column[,...])] ]
... | UNIQUE (column[,...]) [ ENABLED | DISABLED ]
... | CHECK (expression) [ ENABLED | DISABLED ]
}
```

Parameters

CONSTRAINT <i>constraint-name</i>	Assigns a name to the constraint. Vertica recommends that you name all constraints.
PRIMARY KEY	<p>Defines one or more NOT NULL columns as the primary key as follows:</p> <pre>PRIMARY KEY (<i>column</i>[,...]) [ENABLED DISABLED]</pre> <p>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</p> <p>If you do not name a primary key constraint, Vertica assigns the name C_PRIMARY.</p>
FOREIGN KEY	<p>Adds a referential integrity constraint defining one or more columns as foreign keys as follows:</p> <pre>FOREIGN KEY (<i>column</i>[,...]) REFERENCES <i>table</i> [(<i>column</i>[,...])]</pre> <p>If you omit <i>column</i> references, If you omit <i>column</i>, Vertica references the primary key in <i>table</i>.</p>

	<p>If you do not name a foreign key constraint, Vertica assigns the name C_FOREIGN.</p>
UNIQUE	<p>Specifies that the data in a column or group of columns is unique with respect to all table rows, as follows:</p> <pre>UNIQUE (column[,...]) [ENABLED DISABLED]</pre> <p>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</p> <p>If you do not name a unique constraint, Vertica assigns the name C_UNIQUE.</p>
CHECK	<p>Specifies a check condition as an expression that returns a Boolean value, as follows:</p> <pre>CHECK (expression) [ENABLED DISABLED]</pre> <p>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</p> <p>If you do not name a check constraint, Vertica assigns the name C_CHECK.</p>

Privileges

Table owner or user WITH GRANT OPTION is grantor.

- REFERENCES privilege on table to create foreign key constraints that reference this table
- USAGE privilege on schema that contains the table

Enforcing Constraints

A table can specify whether Vertica automatically enforces a primary key, unique key or check constraint with the keyword ENABLED or DISABLED. If you omit ENABLED or DISABLED, Vertica determines whether to enable the constraint automatically by checking the appropriate configuration parameter:

- `EnableNewPrimaryKeysByDefault`
- `EnableNewUniqueKeysByDefault`
- `EnableNewCheckConstraintsByDefault`

For details, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Examples

The following example creates a table (`t01`) with a primary key constraint.

```
CREATE TABLE t01 (id int CONSTRAINT sampleconstraint PRIMARY KEY);
CREATE TABLE
```

This example creates the same table without the constraint, and then adds the constraint with `ALTER TABLE ADD CONSTRAINT`

```
CREATE TABLE t01 (id int);
CREATE TABLE

ALTER TABLE t01 ADD CONSTRAINT sampleconstraint PRIMARY KEY(id);
WARNING 2623: Column "id" definition changed to NOT NULL
ALTER TABLE
```

The following example creates a table (`addapk`) with two columns, adds a third column to the table, and then adds a primary key constraint on the third column.

```
=> CREATE TABLE addapk (col1 INT, col2 INT);
CREATE TABLE

=> ALTER TABLE addapk ADD COLUMN col3 INT;
ALTER TABLE

=> ALTER TABLE addapk ADD CONSTRAINT col3constraint PRIMARY KEY (col3) ENABLED;
WARNING 2623: Column "col3" definition changed to NOT NULL
ALTER TABLE
```

Using the sample table `addapk`, check that the primary key constraint is enabled (`is_enabled is t`).

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_
name IN ('addapk');

constraint_name | column_name | constraint_type | is_enabled
-----+-----+-----+-----
col3constraint | col3       | p               | t
(1 row)
```

This example disables the constraint using `ALTER TABLE ALTER CONSTRAINT`.

```
=> ALTER TABLE addapk ALTER CONSTRAINT col3constraint DISABLED;
```

Check that the primary key is now disabled (`is_enabled` is f).

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_
name IN ('addapk');
```

constraint_name	column_name	constraint_type	is_enabled
col3constraint	col3	p	f

(1 row)

For a general discussion of constraints, see [About Constraints](#). For additional examples of creating and naming constraints, see [Naming Constraints](#).

ALTER USER

Changes a database user account. Changes that you make to a user account affect only future user sessions.

Important: The following ALTER USER parameters are invalid for a user who is added to the Vertica database with the LDAPLink service:

- IDENTIFIED BY
- PROFILE
- SECURITY ALGORITHM

Syntax

```
ALTER USER name user-parameter[,...]
```

user-parameter

```
ACCOUNT { LOCK | UNLOCK }
DEFAULT ROLE roles-expression*
GRACEPERIOD limit
IDENTIFIED BY '[new-password]' [REPLACE 'current-password']
IDLESESSIONTIMEOUT limit
MAXCONNECTIONS limit
MEMORYCAP limit
PASSWORD EXPIRE
PROFILE profile
```

```

RENAME TO new-user-name*
RESOURCE POOL pool-name
RUNTIMECAP expression
SEARCH_PATH path
SECURITY_ALGORITHM 'algorithm'
TEMPSPACECAP limit

```

*** Cannot be used in combination with other parameters**

Parameters

<p><i>name</i></p>	<p>Specifies the name of the new user. Names that contain special characters must be double-quoted. To enforce case-sensitivity, use double-quotes.</p> <p>For details on name requirements, see Creating a Database Name and Password.</p>
<p>ACCOUNT { LOCK UNLOCK }</p>	<p>Locks or unlocks a user's access to the database:</p> <ul style="list-style-type: none"> • UNLOCK (default) • LOCK prevents a new user from logging in. This can be useful when creating an account for a user who does not need immediate access. <p>Tip: To automate account locking, set a maximum number of failed login attempts with CREATE PROFILE.</p>
<p>DEFAULT ROLE <i>roles-expression</i></p>	<p>Specifies what roles are the default roles for this user, with one of the following expressions:</p> <ul style="list-style-type: none"> • NONE (default): Removes all default roles. • A comma-delimited list of roles. • ALL: Sets as default all user roles. • ALL EXCEPT <i>role</i>[, ...] : A comma -delimited list of roles to exclude as default roles. <p>Default roles are automatically activated when a user logs in. The roles specified by this parameter supersede any roles assigned earlier.</p>

	<p>Note: DEFAULT ROLE cannot be specified in combination with other ALTER USER parameters.</p>
GRACEPERIOD <i>Limit</i>	<p>Specifies how long a user query can block on any session socket, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> NONE (default): Removes any grace period previously set on session queries. '<i>interval</i>': Specifies as an interval the maximum grace period for current session queries, up to 20 days. <p>For details, see Handling Session Socket Blocking.</p>
IDENTIFIED BY ' <i>[new-password]</i> ' [REPLACE ' <i>current-password</i> ']	<p>Sets a new password for the user, where <i>new-password</i> must conform to the password complexity policy set by the user's profile.</p> <p>Superusers can change the password for any user, and are not required to specify the REPLACE clause. Non-superusers can only change their own password, and must supply their current password with the REPLACE clause.</p> <p>If you supply an empty string, the user's current password is removed, and the user is no longer prompted for a password when starting a new session.</p> <p>For details, see Password Guidelines and Creating a Database Name and Password.</p>
IDLESESSIONTIMEOUT <i>Limit</i>	<p>The length of time the system waits before disconnecting an idle session, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user. '<i>interval</i>': An interval value, up to one year. <p>For details, see Managing Client Connections.</p>
MAXCONNECTIONS <i>Limit</i>	<p>Indicates the maximum number of connections the user can have to the server, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> NONE (default): No limit set. If you omit this parameter, the user can have an unlimited number of connections.

	<ul style="list-style-type: none"> • <code>'integer'</code> ON NODE: Sets the maximum number of connections to each node to <i>integer</i>. • <code>'integer'</code> ON DATABASE: Sets the maximum number of connections across the database cluster to <i>integer</i>. <p>For details, see Managing Client Connections.</p>
<p>MEMORYCAP <i>Limit</i></p>	<p>Specifies how much memory can be allocated to user requests, where <i>Limit</i> is specified in this format:</p> <ul style="list-style-type: none"> • NONE (default): No limit • <code>'max-expression'</code>: A string value that specifies the memory limit, one of the following: <ul style="list-style-type: none"> ■ <code>int%</code> — Expresses the maximum as a percentage of total memory available to the Resource Manager, where <i>int</i> is an integer value between 0 and 100. For example: <pre>MEMORYCAP '40%'</pre> ■ <code>int{K M G T}</code> — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example: <pre>MEMORYCAP '10G'</pre>
<p>PASSWORD EXPIRE</p>	<p>Forces immediate expiration of the user's password. The user must change the password on the next login.</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: PASSWORD EXPIRE has no effect when using external password authentication methods such as LDAP or Kerberos.</p> </div>
<p>PROFILE <i>profile</i></p>	<p>Assigns the user to a profile, where <i>profile</i> is one of the following:</p> <ul style="list-style-type: none"> • DEFAULT (default): Assigns the default profile to this user. If you omit this parameter, the user is assigned to the default profile. • <code>profile-name</code>: The profile assigned to this user, which includes the user's password policy.

<p>RENAME TO <i>new-user-name</i></p>	<p>Assigns the user a new user name. All privileges assigned to the user remain unchanged.</p> <p>Note: RENAME TO cannot be specified in combination with other ALTER USER parameters.</p>
<p>RESOURCE POOL <i>pool-name</i></p>	<p>Assigns a default resource pool to this user. The user must also be granted privileges to this pool, unless privileges to the pool are set to PUBLIC.</p>
<p>RUNTIMECAP <i>Limit</i></p>	<p>Specifies how long this user's queries can execute, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> • NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user. • '<i>interval</i>' An interval value, up to one year. <p>A query's runtime limit can be set at three levels: the user's runtime limit, the user's resource pool, and the session setting. For more information, see Setting a Runtime Limit for Queries in the Administrator's Guide.</p>
<p>SEARCH_PATH <i>path</i></p>	<p>Specifies the user's default search path that tells Vertica which schemas to search for unqualified references to tables and UDFs, where <i>path</i> is one of the following:</p> <ul style="list-style-type: none"> • DEFAULT (default): Sets the search path as follows: "\$user", public, v_catalog, v_monitor, v_internal • A comma-delimited list of schemas. <p>For details, see Setting Search Paths in the Administrator's Guide.</p>
<p>SECURITY_ ALGORITHM '<i>algorithm</i>'</p>	<p>Set the user-level security algorithm for hash authentication, where <i>algorithm</i> is one of the following:</p> <ul style="list-style-type: none"> • NONE (default): Uses the MD5 algorithm for hash authentication. • MD5 • SHA512

	<p>The user's password expires when you change the SECURITY_ALGORITHM value, and must be reset.</p>
<p>TEMPSPACECAP <i>Limit</i></p>	<p>Limits how much temporary file storage is available for user requests, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> • NONE (default): No limit • '<i>max-expression</i>': A string value that specifies the storage limit, one of the following: <ul style="list-style-type: none"> ■ <i>int</i>% — Expresses storage as a percentage of total file space is available, where <i>int</i> is an integer value between 0 and 100. For example: <pre>TEMPSPACECAP '40%'</pre> ■ <i>int</i>{K M G T} — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example: <pre>TEMPSPACECAP '10G'</pre>

Privileges

- Superuser
- Non-superusers can change their own user accounts with these options:
 - IDENTIFIED BY
 - RESOURCE POOL
 - SEARCH_PATH
 - SECURITY_ALGORITHM

Examples

Change the user password

```
=> CREATE USER user1;  
=> ALTER USER user1 IDENTIFIED BY 'newpassword';
```

Change the security algorithm and password

This example changes the user's hash authentication and password to SHA-512 and newpassword, respectively. When you execute the ALTER USER statement, Vertica hashes the password, using the SHA-512 algorithm, and saves the hashed version:

```
=> CREATE USER user1;  
=> ALTER USER user1 SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';
```

Assign user default roles

This example makes a user's assigned roles the user's default roles. The first ALTER USER statement makes role1 the default role. The second ALTER USER statement makes role1, role2, and role3 the user's default roles.

```
=> CREATE USER user1;  
CREATE USER  
=> GRANT role1, role2, role3 to user1;  
=> ALTER USER user1 default role role1;  
=> ALTER USER user1 default role ALL;
```

Assign user default roles with EXCEPT

This example makes all the user's assigned roles default roles with the exception of role1.

```
=> GRANT role1, role2, role3 to user1;  
=> ALTER USER user1 default role ALL EXCEPT r1;
```

See Also

- [CREATE USER](#)
- [DROP USER](#)

ALTER VIEW

Modifies the metadata of an existing view. The changes are auto-committed

Syntax

General Usage

```
ALTER VIEW [schema.]view-name {
... | OWNER TO owner-name
... | SET SCHEMA schema-name
... | { INCLUDE | EXCLUDE | MATERIALIZE } [ SCHEMA ] PRIVILEGES
```

View Renaming

```
ALTER TABLE [schema.]view-name[,...] RENAME TO new-view-name[,...]
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDbObject</pre>
<i>view-name</i>	The view to alter.
SET SCHEMA <i>schema-name</i>	Moves the view from one schema to another.
OWNER TO <i>owner-name</i>	Changes the view owner.
{ INCLUDE EXCLUDE MATERIALIZE } [SCHEMA] PRIVILEGES	<p>Specifies default inheritance of schema privileges for this view:</p> <ul style="list-style-type: none"> EXCLUDE [SCHEMA] PRIVILEGES (default) disables inheritance of privileges from the schema. INCLUDE [SCHEMA] PRIVILEGES grants the view the same privileges granted to its schema. MATERIALIZE: Copies grants to the view and creates a GRANT object on the view. This disables the inherited privileges flag on the view, so you can: <ul style="list-style-type: none"> Grant more specific privileges at the view level Use schema-level privileges as a template

	<ul style="list-style-type: none">■ Move the view to a different schema■ Change schema privileges without affecting the view <p>Note: If inherited privileges are disabled at the database level, schema privileges can still be materialized.</p> <p>For more information see Grant Inherited Privileges.</p>
RENAME TO	<p>Renames one or more views:</p> <pre>RENAME TO <i>new-view-name</i>[,...]</pre> <p>The following requirements apply:</p> <ul style="list-style-type: none">• The new view name must be unique among all tables, views and projections within the same schema.• If you specify multiple views to rename, the source and target lists must have the same number of names.• Renaming a view requires USAGE and CREATE privileges on the schema that contains the view.

Privileges

Changing a view requires the following privileges:

- Superuser
- If renaming a view, CREATE privileges on the schema in which the view is renamed.

Example

The following command renames view1 to view2:

```
=> CREATE VIEW view1 AS SELECT * FROM t;  
CREATE VIEW  
=> ALTER VIEW view1 RENAME TO view2;  
ALTER VIEW
```

BEGIN

Starts a transaction block. BEGIN is a synonym for [START TRANSACTION](#).

Syntax

```
BEGIN [ WORK | TRANSACTION ] [ isolation-level ] [ transaction-mode ]
```

and where *transaction_mode* is one of:

```
READ { ONLY | WRITE }
```

Parameters

WORK TRANSACTION	Optional keywords for readability purposes only.
<i>isolation-level</i>	<p>Specifies the transaction's isolation level, which determines what data the transaction can access when other transactions are running concurrently. You can set <i>isolation-level</i> to one of the following:</p> <ul style="list-style-type: none">• READ COMMITTED (default)• SERIALIZABLE• REPEATABLE READ (automatically converted to SERIALIZABLE)• READ UNCOMMITTED (automatically converted to READ COMMITTED) <p>For detailed information, see Transactions in Vertica Concepts.</p>
<i>transaction-mode</i>	
READ { ONLY WRITE }	<p>Transaction mode can be one of the following:</p> <ul style="list-style-type: none">• READ WRITE—(default)The transaction is read/write.• READ ONLY—The transaction is read-only.

Setting the transaction session mode to read-only disallows the following SQL commands, but does not prevent all disk write operations:

- INSERT, UPDATE, DELETE, and COPY if the table they would write to is not a temporary table
- All CREATE, ALTER, and DROP commands
- GRANT, REVOKE, and EXPLAIN if the command it would run is among those listed.

Privileges

None

Examples

This example shows how to begin a transaction and set the isolation level.

```
=> BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;  
BEGIN  
  
=> CREATE TABLE sample_table (a INT);  
CREATE TABLE  
  
=> INSERT INTO sample_table (a) VALUES (1);  
OUTPUT  
-----  
1  
(1 row)
```

See Also

- [Transactions](#)
- [Creating Transactions](#)
- [COMMIT](#)

- [END](#)
- [ROLLBACK](#)

COMMENT ON Statements

The following functions allow you to create comments associated with Vertica database objects:

- [COMMENT ON COLUMN](#)
- [COMMENT ON CONSTRAINT](#)
- [COMMENT ON FUNCTION](#)
- [COMMENT ON LIBRARY](#)
- [COMMENT ON NODE](#)
- [COMMENT ON PROJECTION](#)
- [COMMENT ON SCHEMA](#)
- [COMMENT ON SEQUENCE](#)
- [COMMENT ON TABLE](#)
- [COMMENT ON TRANSFORM FUNCTION](#)
- [COMMENT ON VIEW](#)

COMMENT ON COLUMN

Adds, revises, or removes a projection column comment. You can only add comments to projection columns, not to table columns. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON COLUMN [schema.]proj-name.column-name IS {'comment' | NULL}
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDbObject</pre>
<i>proj-name.column-name</i>	<p>Specifies the name of the projection and column with which to associate the comment.</p>
<i>comment</i>	<p>Specifies the comment text to add. If a comment already exists for this column, this comment overwrites the previous comment.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>You can enclose a blank value within single quotes to remove an existing comment.</p>
NULL	<p>Removes an existing comment.</p>

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Example

The following example adds a comment to the `customer_name` column in the `customer_dimension` projection:

```
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS 'Last name only';
```

The following examples remove a comment from the `customer_name` column in the `customer_dimension` projection in two ways, using the NULL option, or specifying a blank string:

```
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS NULL;  
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS '';
```

COMMENT ON CONSTRAINT

Adds, revises, or removes a comment on a constraint. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON CONSTRAINT constraint-name ON [ [db-name.]schema.]table-name IS ... {'comment' | NULL };
```

Parameters

<i>constraint-name</i>	The name of the constraint associated with the comment.
[<i>db-name</i> .] <i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>table-name</i>	Specifies the name of the table constraint with which to associate a comment.
<i>comment</i>	Specifies the comment text to add. If a comment already exists for this constraint, this comment overwrites the previous one. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. You can enclose a blank value within single quotes to remove an existing comment.
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Example

The following example adds a comment to the `constraint_x` constraint on the `promotion_dimension` table:

```
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS 'Primary key';
```

The following examples remove a comment from the `constraint_x` constraint on the `promotion_dimension` table:

```
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS NULL;  
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS '';
```

COMMENT ON FUNCTION

Adds, revises, or removes a comment on a function. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON FUNCTION [schema.]function-name function-arg IS { 'comment' | NULL };
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
---------------	---

<i>function-name</i>	Specifies the name of the function with which to associate the comment.
<i>function-arg</i>	Specifies the function arguments.
<i>comment</i>	<p>Specifies the comment text to add. If a comment already exists for this function, this overwrites the previous one.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>Enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the `macros.zerowhennull (x INT)` function:

```
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS 'Returns a 0 if not NULL';
```

The following examples remove a comment from the `macros.zerowhennull (x INT)` function in two ways by using the NULL option, or specifying a blank string:

```
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS NULL;  
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS '';
```

COMMENT ON LIBRARY

Adds, revises, or removes a comment on a library . Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON LIBRARY [ schema.]library-name IS {'comment' | NULL}
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>library-name</i>	The name of the library associated with the comment.
<i>comment</i>	<p>Specifies the comment text to add. If a comment already exists for this library, this comment overwrites the previous one.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>Enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the library MyFunctions :

```
=> COMMENT ON LIBRARY MyFunctions IS 'In development';
```

The following examples remove a comment from the library MyFunctions :

```
=> COMMENT ON LIBRARY MyFunctions IS NULL;  
=> COMMENT ON LIBRARY MyFunctions IS '';
```

See Also

- [COMMENTS](#)

COMMENT ON NODE

Adds, revises, or removes a comment on a node. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Dropping an object drops all comments associated with the object.

Syntax

```
COMMENT ON NODE node-name IS { 'comment' | NULL }
```

Parameters

<i>node-name</i>	The name of the node associated with the comment.
<i>comment</i>	<p>Specifies the comment text to add. If a comment already exists for this node, this comment overwrites the previous one.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>Enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.

- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment for the `initiator` node:

```
=> COMMENT ON NODE initiator IS 'Initiator node';
```

The following examples removes a comment from the `initiator` node.

```
=> COMMENT ON NODE initiator IS NULL;  
=> COMMENT ON NODE initiator IS '';
```

See Also

[COMMENTS](#)

COMMENT ON PROJECTION

Adds, revises, or removes a comment on a projection. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Dropping an object drops all comments associated with the object.

Syntax

```
COMMENT ON PROJECTION [ schema.]projection IS { 'comment' | NULL }
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>projection</i>	The name of the projection associated with the comment.
<i>comment</i>	Specifies the text of the comment to add. If a comment already exists for this projection, the comment you enter here overwrites the previous

	<p>comment.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>Enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the `customer_dimension_vmart_node01` projection:

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS 'Test data';
```

The following examples remove a comment from the `customer_dimension_vmart_node01` projection:

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS NULL;  
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS '';
```

See Also

[COMMENTS](#)

COMMENT ON SCHEMA

Adds, revises, or removes a comment on a schema. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON SCHEMA schema-name IS {'comment' | NULL}
```

Parameters

<i>schema-name</i>	The schema associated with the comment.
<i>comment</i>	<p>Text of the comment to add. If a comment already exists for this schema, the comment you enter here overwrites the previous comment.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>You can enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the `public` schema:

```
=> COMMENT ON SCHEMA public IS 'All users can access this schema';
```

The following examples remove a comment from the `public` schema.

```
=> COMMENT ON SCHEMA public IS NULL;  
=> COMMENT ON SCHEMA public IS '';
```

COMMENT ON SEQUENCE

Adds, revises, or removes a comment on a sequence. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON SEQUENCE [schema.]sequence-name IS { 'comment' | NULL }
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>sequence-name</i>	The name of the sequence associated with the comment.
<i>comment</i>	<p>Specifies the text of the comment to add. If a comment already exists for this sequence, this comment overwrites the previous one.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>You can enclose a blank value within single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the sequence called `prom_seq`.

```
=> COMMENT ON SEQUENCE prom_seq IS 'Promotion codes';
```

The following examples remove a comment from the `prom_seq` sequence.

```
=> COMMENT ON SEQUENCE prom_seq IS NULL;  
=> COMMENT ON SEQUENCE prom_seq IS '';
```

COMMENT ON TABLE

Adds, revises, or removes a comment on a table. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON TABLE [ schema.]table-name IS { 'comment' | NULL }
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>table-name</i>	<p>Specifies the name of the table with which to associate the comment.</p>
<i>comment</i>	<p>Specifies the text of the comment to add. Enclose the text of the comment within single-quotes. If a comment already exists for this table, the comment you enter here overwrites the previous comment.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>You can enclose a blank value within single quotes to remove an existing comment.</p>

Null	Removes a previously added comment.
------	-------------------------------------

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments

Examples

The following example adds a comment to the `promotion_dimension` table:

```
=> COMMENT ON TABLE promotion_dimension IS '2011 Promotions';
```

The following examples remove a comment from the `promotion_dimension` table:

```
=> COMMENT ON TABLE promotion_dimension IS NULL;  
=> COMMENT ON TABLE promotion_dimension IS '';
```

See Also

- [COMMENTS](#)

COMMENT ON TRANSFORM FUNCTION

Adds, revises, or removes a comment on a user-defined transform function. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON TRANSFORM FUNCTION [schema.]tfunction-name  
...( [ tfunction-arg-name tfunction-arg-type ][,...] ) IS {'comment' | NULL}
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>tfunction-name</i>	<p>Specifies name of the transform function with which to associate the comment.</p>
<i>tfunction-arg-name</i> <i>tfunction-arg-type</i>	<p>Indicates the names and data types of one or more transform function arguments. If you supply argument names and types, each type must match the type specified in the library used to create the original transform function.</p>
<i>comment</i>	<p>Specifies the comment text to add. If a comment already exists for this transform function, this comment overwrites the previous one.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>Enclose a blank value within single quotes to remove an existing comment.</p>
NULL	<p>Removes an existing comment.</p>

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the `macros.zerowhennull (x INT) UTF` function:

```
=> COMMENT ON TRANSFORM FUNCTION macros.zerowhennull(x INT) IS 'Returns a 0 if not NULL';
```

The following example removes a comment from the `macros.zerowhennull (x INT)` function by using the `NULL` option:

```
=> COMMENT ON TRANSFORM FUNCTION macros.zerowhennull(x INT) IS NULL;
```

COMMENT ON VIEW

Adds, revises, or removes a comment on a view. Each object can have one comment. Comments are stored in the system table [COMMENTS](#).

Syntax

```
COMMENT ON VIEW [schema.]view-name IS { 'comment' | NULL }
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>view-name</i>	The name of the view with which to associate the comment.
<i>comment</i>	<p>Specifies the text of the comment to add. If a comment already exists for this view, this comment overwrites the previous one.</p> <p>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</p> <p>Enclose a blank value in single quotes to remove an existing comment.</p>
NULL	Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to a view called `curr_month_ship`:

```
=> COMMENT ON VIEW curr_month_ship IS 'Shipping data for the current month';
```

The following example removes a comment from the `curr_month_ship` view:

```
=> COMMENT ON VIEW curr_month_ship IS NULL;
```

COMMIT

Ends the current transaction and makes all changes that occurred during the transaction permanent and visible to other users.

COMMIT is a synonym for [END](#)

Syntax

```
COMMIT [ WORK | TRANSACTION ]
```

Parameters

WORK TRANSACTION	Have no effect; they are optional keywords for readability.
--------------------	---

Privileges

None

Examples

This example shows how to commit an insert.

```
=> CREATE TABLE sample_table (a INT);
=> INSERT INTO sample_table (a) VALUES (1);
OUTPUT
-----
1
=> COMMIT;
```

See Also

- [Transactions](#)
- [Creating Transactions](#)
- [BEGIN](#)
- [ROLLBACK](#)
- [START TRANSACTION](#)

CONNECT

Connects to another Vertica database to enable data import (using the [COPY FROM VERTICA](#) statement) or export (using the [EXPORT](#) statement). By default, invoking CONNECT occurs over the Vertica private network. Creating a connection over a public network requires some configuration. For information about using CONNECT to export data to or import data over a public network, see [Using Public and Private IP Networks](#).

When importing from or exporting to a Vertica database, you can connect only to a database that uses trusted- (username-only) or password-based authentication, as described in [Security and Authentication](#). SSL authentication is not supported.

Syntax

```
CONNECT TO VERTICA database USER username PASSWORD 'password' ON 'host',port
```

Parameters

<i>database</i>	The connection target database name.
<i>username</i>	The username to use when connecting to the other database.
<i>password</i>	A string containing the password to use to connect to the other database.
<i>host</i>	A string containing the host name of one of the nodes in the other database.
<i>port</i>	The port number of the other database as an integer.

Privileges

No special permissions required.

Connection Details

Once you successfully establish a connection to another database, the connection remains open for the current session. To disconnect a connection, use the [DISCONNECT](#) statement.

You can have only one connection to another database at a time, though you can create connections to multiple different databases in the same session.

If the target database does not have a password, and you specify a password in the `CONNECT` statement, the connection succeeds, but does not give any indication that you supplied an incorrect password.

Example

```
=> CONNECT TO VERTICA ExampleDB USER dbadmin PASSWORD 'Password123' ON 'VerticaHost01',5433;  
CONNECT
```

See Also

- [COPY FROM VERTICA](#)
- [DISCONNECT](#)
- [EXPORT TO VERTICA](#)

COPY

COPY bulk-loads data into a Vertica database. By default, COPY automatically commits itself and any current transaction except when loading temporary tables. If COPY is terminated or interrupted Vertica rolls it back.

For information on loading one or more files or pipes on a cluster host or on a client system, see [COPY LOCAL](#).

Syntax

```
COPY [schema-name.]target-table
... [ ( { column-as-expression | column }
..... [ DELIMITER [ AS ] 'char' ]
..... [ ENCLOSED [ BY ] 'char' ]
..... [ ENFORCELENGTH ]
..... [ ESCAPE [ AS ] 'char' | NO ESCAPE ]
..... [ FILLER datatype]
..... [ FORMAT 'format' ]
..... [ NULL [ AS ] 'string' ]
..... [ TRIM 'byte' ]
... [, ... ] ) ]
... [ COLUMN OPTION ( column
..... [ DELIMITER [ AS ] 'char' ]
..... [ ENCLOSED [ BY ] 'char' ]
..... [ ENFORCELENGTH ]
..... [ ESCAPE [ AS ] 'char' | NO ESCAPE ]
..... [ FORMAT 'format' ]
..... [ NULL [ AS ] 'string' ]
..... [ TRIM 'byte' ]
... [, ... ] ) ]
[ FROM {
...STDIN
..... [ input-file-format ]
...| 'pathToData' [ ON nodename | ON (nodeset) | ON ANY NODE ]
..... [ input-file-format ] [, ...]
...| LOCAL {STDIN | 'pathToData'}
..... [ input-file-format ] [, ...]
```

```
...| VERTICA source_database. [source_schema]. source_table [(source_column [,...]) ]  
} ]  
  
...[ parser ]  
...[ WITH ] UDL-clause[...]  
...[ DELIMITER [ AS ] 'char' ]  
...[ TRAILING NULLCOLS ]  
...[ NULL [ AS ] 'string' ]  
...[ ESCAPE [ AS ] 'char' | NO ESCAPE ]  
...[ ENCLOSED [ BY ] 'char' ]  
...[ RECORD TERMINATOR 'string' ]  
...[ SKIP records ]  
...[ SKIP BYTES integer ]  
...[ TRIM 'byte' ]  
...[ REJECTMAX integer ]  
...[ REJECTED DATA {'path' [ ON nodename ] [, ...] | AS TABLE reject-table} ]  
...[ EXCEPTIONS 'path' [ ON nodename ] [, ...] ]  
...[ ENFORCELENGTH ]  
...[ ERROR TOLERANCE ]  
...[ ABORT ON ERROR ]  
...[ [ STORAGE ] load-method ]  
...[ STREAM NAME 'streamName' ]  
...[ NO COMMIT ]
```

Parameters

See [COPY Parameters](#)

Privileges

General

Superusers have full COPY privileges. The following requirements apply to non-superusers:

- USER-accessible storage location
- Applicable READ or WRITE privileges granted to the storage location where files are read or written

COPY LOCAL

- INSERT privileges to copy data from the STDIN pipe
- USAGE privileges on the schema

COPY FROM STDIN

- INSERT privilege on table
- USAGE privilege on schema

File Paths for Rejected Data and Exceptions

COPY can specify a path to store rejected data and exceptions. If the path resolves to a storage location, the following privileges apply to non-superusers:

- The storage location was created with the USER option (see [CREATE LOCATION](#)).
- The user must have READ access to the storage location, as described in [GRANT \(Storage Location\)](#)

COPY Topics

[COPY Option Parser Dependencies](#)

[COPY Restrictions](#)

[Setting vsql Variables](#)

[See Also](#)

COPY Parameters

Note: For details on which COPY parameters are valid for specific parsers, see [COPY Option Parser Dependencies](#).

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre> <p>COPY ignores <i>schema-name</i> when used in CREATE EXTERNAL TABLE or CREATE FLEX EXTERNAL TABLE statements.</p>
<i>target-table</i>	<p>The target columnar or flexible table for loading new data. Vertica loads the data into all projections that include columns from the schema table.</p>

<p><i>column-as-expression</i></p>	<p>Specifies the expression used to compute values for the target column. For example:</p> <pre>COPY t(year AS TO_CHAR(k, 'YYYY')) FROM 'myfile.dat'</pre> <p>Use this option to transform data when it is loaded into the target database.</p> <p>For details about:</p> <ul style="list-style-type: none">• Expressions with COPY: see Transforming Data During Loads in the Administrator's Guide.• Fillers: see Manipulating Source Data Columns in the Administrator's Guide.
<p><i>column</i></p>	<p>Restricts the load to one or more specified columns in the table. If you do not specify any columns, COPY loads all columns by default.</p> <p>Table columns that you do not specify in the column list are assigned their default values. If a column had no defined default value, COPY inserts NULL.</p> <p>If you leave the <code>column</code> parameter blank to load all columns in the table, you can use the optional parameter <code>COLUMN OPTION</code> to specify parsing options for specific columns.</p> <p>The data file must contain the same number of columns as the COPY command's column list. For example, in a table T1 with nine columns (C1 through C9), the following statement loads the three columns of data in each record to columns C1, C6, and C9, respectively:</p> <pre>=> COPY T1 (C1, C6, C9);</pre>
<p>FILLER <i>datatype</i></p>	<p>Specifies not to load the column and its fields into the destination table. Use this option to omit columns that you do not want to transfer into a table.</p> <p>This parameter also transforms data from a source column and loads the transformed data to the destination table, rather than loading the original, untransformed source column (parsed column).</p>

	<p>Note: If <i>datatype</i> is VARCHAR, set the VARCHAR length (VARCHAR(<i>n</i>)) so the combined length of all FILLER source fields does not exceed the target column's defined length. Otherwise, the COPY command can return with an error.</p> <p>For more information, see Manipulating Source Data Columns in the Administrator's Guide</p>
<p>FORMAT '<i>format</i>'</p>	<p>Specifies the input formats to use when loading date/time and binary columns, where <i>format</i> can be one of the following:</p> <p>These are the valid input formats when loading binary columns:</p> <ul style="list-style-type: none"> • octal • hex • bitstream <p>See Loading Binary (Native) Data to learn more about these formats.</p> <p>When loading date/time columns, using FORMAT significantly improves load performance. COPY supports the same formats as the TO_DATE function.</p> <p>See the following topics for additional information:</p> <ul style="list-style-type: none"> • Template Patterns for Date/Time Formatting • Template Pattern Modifiers for Date/Time Formatting <p>If you specify invalid format strings, the COPY operation returns an error.</p>
<p><i>pathToData</i></p>	<p>Specifies the absolute path of the file (or files) containing the data, which can be from multiple input sources. If the file is stored in HDFS, <i>pathToData</i> is a URL in the hdfs scheme, typically 'hdfs:///path/to/file'. See Reading Directly from HDFS.</p> <p><i>pathToData</i> can optionally contain wildcards to match more than one file. The file or files must be accessible to the local</p>

	<p>client or the host on which the COPY statement runs. COPY skips empty files in the file list. A file list that includes directories causes the query to fail. See Specifying COPY FROM Options. The supported patterns for wildcards are specified in the Linux Manual Page for Glob (7), and for ADO.net platforms, through the .NET Directory.GetFiles Method.</p> <p>You can use variables to construct the pathname as described in Using Load Scripts.</p> <p>If <i>path</i> resolves to a storage location on a local file system, and the user invoking COPY is not a superuser, the following privileges apply:</p> <ul style="list-style-type: none"> • The storage location must have been created with the USER option (see CREATE LOCATION). • The user must already have been granted READ access to the file storage location, as described in GRANT (Storage Location) <p>Further, if a user has privileges but is not a superuser, and invokes COPY from that storage location, Vertica ensures that symbolic links do not result in unauthorized access.</p>
<p>ON <i>nodename</i></p>	<p>Specifies the node on which the data to copy resides and the node that should parse the load file. If you omit <i>nodename</i>, the location of the input file defaults to the COPY initiator node. Use <i>nodename</i> to copy and parse a load file from a node other than the COPY initiator node.</p> <div data-bbox="597 1377 1403 1528" style="background-color: #f0f0f0; padding: 10px;"> <p>Note: <i>nodename</i> is invalid with STDIN and LOCAL: STDIN is read on the initiator node only, and LOCAL indicates a client node.</p> </div>
<p>ON (<i>nodeset</i>)</p>	<p>Specifies a set of nodes on which to perform the load. The same data must be available for load on all named nodes. <i>nodeset</i> is a comma-separated list of node names in parentheses. For example:</p> <div data-bbox="597 1740 1403 1829" style="background-color: #f0f0f0; padding: 10px;"> <pre>=> COPY t FROM 'file1.txt' ON (v_vmart_node0001, v_vmart_node0002);</pre> </div> <p>Vertica apportions the load among all of the specified nodes. If</p>

	<p>you also specify <code>ERROR TOLERANCE</code> or <code>REJECTMAX</code>, Vertica instead chooses a single node on which to perform the load.</p> <p>If the data is available on all nodes, you usually use <code>ON ANY NODE</code>. However, you can use <code>ON <i>nodeset</i></code> to do manual load-balancing among concurrent loads.</p>
<code>ON ANY NODE</code>	<p>Specifies that the source file to load is available on all of the nodes, so <code>COPY</code> opens the file and parses it from any node(s) in the cluster.</p> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <p>Caution: The file must be the same on all nodes. If the file differs on two nodes, an incorrect or incomplete result is returned, with no error or warning.</p> </div> <p>Vertica attempts to apportion the load among several nodes if the file is large enough to benefit from apportioning. It chooses a single node if <code>ERROR TOLERANCE</code> or <code>REJECTMAX</code> is specified.</p> <p>You can use a wildcard or glob (such as <code>*.dat</code>) to load multiple input files, combined with the <code>ON ANY NODE</code> clause. If you use a glob, <code>COPY</code> distributes the list of files to all cluster nodes and spreads the workload.</p> <p><code>ON ANY NODE</code> is invalid with <code>STDIN</code> and <code>LOCAL: STDIN</code> can only use the initiator node, and <code>LOCAL</code> indicates a client node.</p> <p><code>ON ANY NODE</code> is the default for HDFS paths and does not need to be specified.</p>
<code>STDIN</code>	<p>Reads from the client a standard input instead of a file. <code>STDIN</code> takes one input source only and is read on the initiator node. To load multiple input sources, use <i>pathToData</i>.</p> <p>User must have <code>INSERT</code> privileges on the table and <code>USAGE</code> privileges on its schema.</p>
<code>LOCAL {STDIN '<i>pathToData</i>'}</code>	<p>Specifies that all paths for the <code>COPY</code> statement are on the client system and that all <code>COPY</code> variants are initiated from a client. You can use <code>LOCAL</code> and <i>pathtodata</i> (see above), to specify a relative path.</p> <p>For details, see COPY LOCAL.</p>
<i>input-file-format</i>	<p>Specifies the input file format, one of the following:</p>

	<ul style="list-style-type: none">• UNCOMPRESSED (default)• BZIP• GZIP• LZO <p>Input files can be of any format. If you use wildcards, all qualifying input files must be in the same format. To load different file formats, specify the format types specifically.</p> <p>The following requirements and restrictions apply:</p> <ul style="list-style-type: none">• When using concatenated BZIP or GZIP files, verify that all source files terminate with a record terminator before concatenating them.• Concatenated BZIP and GZIP files are not supported for NATIVE (binary) and NATIVE VARCHAR formats.• LZO files are assumed to be compressed with lzop. Supported lzop arguments are: -F, --no-checksum, --crc32, --adler32, -n, --no-name, -N, --name, --no-mode, --no-time, --fast, --best, and the numbered compression levels. See lzop.org for details.• BZIP, GZIP, and LZO compression cannot be used with ORC format.
<i>parser</i>	<p>Specifies the parser to use when bulk loading columnar tables, one of the following:</p> <ul style="list-style-type: none">• NATIVE• NATIVE VARCHAR• FIXEDWIDTH• ORC[(hive_partition_cols='partitions')]• PARQUET[(hive_partition_cols='partitions')] <p>By default, COPY uses the DELIMITER parser for UTF-8 format, delimited text input data.</p>

	<p>To use a flex table parser for column tables, use the <code>PARSER</code> parameter followed by a flex table parser argument. For parser descriptions, see Flex Parsers Reference.</p> <p>Note: You do not specify the <code>DELIMITER</code> parser directly; absence of a specific parser indicates the default.</p> <p>For more information, see Specifying a COPY Parser in the Administrator's Guide.</p> <p>The following restrictions apply:</p> <ul style="list-style-type: none"> • These parsers are not applicable when loading flexible tables. • The <code>ORC</code> and <code>PARQUET</code> parsers are for use with Hadoop files in those formats. The files do not need to be stored in HDFS. For details, see Reading Hadoop Columnar File Formats. • <code>COPY LOCAL</code> does not support <code>NATIVE</code> and <code>NATIVE VARCHAR</code> parsers. • To use a flex table parser for column tables, use the <code>PARSER</code> parameter followed by a flex table parser argument. For supported flex table parsers, see Bulk Loading Data into Flex Tables.
<p>[WITH] <i>UDL-clause</i> [...]</p>	<p>Alone, improve readability of the statement. Using <code>WITH</code> has no effect on the actions performed by the statement.</p> <p>Followed by a UDL-clause, specifies one or more user-defined load functions—one source, one or more filters, and one parser, as follows:</p> <ul style="list-style-type: none"> • <code>SOURCE</code> <i>source</i>([arg=value[,...]]) • <code>FILTER</code> <i>filter</i>([arg=value[,...]]) • <code>PARSER</code> <i>parser</i>([arg=value[,...]]) <p>To use a flex table parser for column tables, use the <code>PARSER</code> parameter followed by a flex table parser argument. For supported flex table parsers, see Bulk Loading Data into</p>

	Flex Tables .
COLUMN OPTION	Specifies load metadata for one or more columns declared in the table column list. For example, you can specify that a column has its own DELIMITER, ENCLOSED BY, NULL as 'NULL' expression, and so on. You do not have to specify every column name explicitly in the COLUMN OPTION list, but each column you specify must correspond to a column in the table column list.
COLSIZES (<i>integer</i> [,...])	Specifies column widths when loading fixed-width data. COPY requires that you specify the COLSIZES when using the FIXEDWIDTH parser. COLSIZES and the list of integers must correspond to the columns listed in the table column list. For more information, see Loading Fixed-Width Format Data in the Administrator's Guide.
DELIMITER	Indicates the single ASCII character used to separate columns within each record of a file. You can use any ASCII value in the range E '\000' to E '\177', inclusive. You cannot use the same character for both the DELIMITER and NULL parameters. For more information, see Loading UTF-8 Format Data in the Administrator's Guide. Default: Vertical bar ().
TRAILING NULLCOLS	Specifies that if Vertica encounters a record with insufficient data to match the columns in the table column list, COPY inserts the missing columns with NULLs. For other information and examples, see Loading Fixed-Width Format Data in the Administrator's Guide.
ESCAPE [AS]	Sets the escape character. Once set, the character following the escape character is interpreted literally, rather than as a special character. You can define an escape character using any ASCII value in the range E '\001' to E '\177', inclusive (any ASCII character except NULL: E '\000'). Note that the backslash character ('\') is the default escape character. The COPY statement does not interpret the data it reads in as String Literals . It also does not follow the same escape rules as other SQL statements (including the COPY parameters). When reading in data, COPY interprets only the characters defined by

	<p>these options as special values:</p> <ul style="list-style-type: none"> • ESCAPE [AS] • DELIMITER • ENCLOSED [BY] • RECORD TERMINATOR
NO ESCAPE	Eliminates escape character handling. Use this option if you do not need any escape character and you want to prevent characters in your data from being interpreted as escape sequences.
ENCLOSED [BY]	Sets the quote character within which to enclose data, allowing delimiter characters to be embedded in string values. You can choose any ASCII value in the range E '\001' to E '\177' inclusive (any ASCII character except NULL: E '\000'). By default, ENCLOSED BY has no value, meaning data is not enclosed by any sort of quote character.
NULL	The string representing a null value. The default is an empty string (' '). You can specify a null value as any ASCII value in the range E '\001' to E '\177' inclusive (any ASCII character except NULL: E '\000'). You cannot use the same character for both the DELIMITER and NULL options. For more information, see Loading UTF-8 Format Data .
RECORD TERMINATOR	Specifies the literal character string indicating the end of a data file record. For more information about using this parameter, see Loading UTF-8 Format Data .
SKIP <i>records</i>	Indicates the number (integer) of records to skip in a load file. For example, you can use the SKIP option to omit table header information.
SKIP BYTES <i>total</i>	Indicates the <i>total</i> number (integer) of bytes in a record to skip. This option is only available when loading fixed-width data.
TRIM	Trims the number of bytes you specify from a column. This option is only available when loading fixed-width data. You can

	<p>set TRIM at the table level for a column, or as part of the COLUMN OPTION parameter.</p>
REJECTMAX	<p>Specifies a maximum number of logical records that can be rejected before a load fails. For details, see Capturing Load Rejections and Exceptions.</p> <p>REJECTMAX disables apportioned load.</p>
REJECTED DATA	<p>Specifies to write each row that failed to load due to a parsing error to the specified target, as follows:</p> <pre>REJECTED DATA { 'path' [ON nodename] [, ...] AS TABLE reject-table }</pre> <p>Vertica can write rejected data to the specified path or to a table:</p> <ul style="list-style-type: none"> 'path' [ON nodename]: Copies the rejected row data to the specified path on the node executing the load. If qualified by ON nodename, Vertica moves existing rejected data files on nodename to path on the same node. REJECTED DATA AS TABLE reject-table: Saves rejected rows to columnar table reject-table. <p>When this parameter is used with COPY...ON ANY NODE, rejected data processed by each node is written to path on that node. To collect all rejected data in one place regardless of how the load is distributed, use a table.</p> <p>For details about both options, see Capturing Load Rejections and Exceptions in the Administrator's Guide.</p>
EXCEPTIONS	<p>Specifies the file name or absolute path of the file in which to write exceptions, as follows:</p> <pre>EXCEPTIONS 'path' [ON nodename] [,...]</pre> <p>Files are written on the node or nodes executing the load</p> <p>Exceptions describe why each rejected row was rejected. Each exception describes the corresponding record in the file specified by the REJECTED DATA option.</p> <p>If path resolves to a storage location, the following privileges</p>

	<p>apply to non-superusers:</p> <ul style="list-style-type: none"> • The storage location must be created with the USER option (see CREATE LOCATION). • The user must have READ access to the storage location where the files exist, as described in GRANT (Storage Location). <p>The ON <i>nodename</i> clause moves existing exceptions files on <i>nodename</i> to the indicated <i>path</i> on the same node. For details, see Saving Load Exceptions (EXCEPTIONS) in the Administrator's Guide.</p> <p>When this parameter is used with COPY . . . ON ANY NODE, exceptions processed by each node are written to <i>path</i> on that node.</p> <p>Specifying an exceptions file name is incompatible with the REJECTED DATA AS TABLE clause. Exceptions are listed in the table's rejected_reason column.</p>
ENFORCELENGTH	<p>Determines whether COPY truncates or rejects data rows of type char, varchar, binary, and varbinary if they do not fit the target table. Specifying the optional ENFORCELENGTH parameter rejects rows.</p> <p>You can set ENFORCELENGTH at the table level for a column, or as part of the COLUMN OPTION parameter.</p> <p>Default: COPY truncates offending rows of these data types, but does not reject them. For more details, see Tracking Load Exceptions and Rejections Status in the Administrator's Guide.</p>
ERROR TOLERANCE	<p>Specifies that COPY treats each source during execution independently when loading data. The statement is not rolled back if a single source is invalid. The invalid source is skipped and the load continues.</p> <p>This parameter is disabled for ORC and Parquet files; specifying it has no effect.</p> <p>Using this parameter disables apportioned load.</p>
ABORT ON ERROR	<p>Specifies that COPY stops if any row is rejected. The statement is rolled back and no data is loaded.</p>

<p>[STORAGE] <i>Load-method</i></p>	<p>Specifies how to load data into the database, one of the following:</p> <ul style="list-style-type: none"> • AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads. • DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads. • TRICKLE: Loads data only into WOS, suitable for frequent incremental loads. <p>This option is invalid for external tables.</p> <p>For details, see Choosing a Load Method in the Administrator's Guide.</p>
<p>STREAM NAME</p>	<p>[Optional] Supplies a COPY load stream identifier. Using a stream name helps to quickly identify a particular load. The STREAM NAME value that you supply in the load statement appears in the stream column of the LOAD_STREAMS system table.</p> <p>A valid stream name can contain any combination of alphanumeric or special characters up to 128 bytes in length.</p> <p>By default, Vertica names streams by table and file name. For example, if you are loading two files (f1, f2) into TableA, their default stream names are TableA-f1, TableA-f2, respectively.</p> <p>To name a stream: => <code>COPY mytable FROM myfile DELIMITER ' ' DIRECT STREAM NAME 'My stream name';</code></p>
<p>NO COMMIT</p>	<p>Prevents the COPY statement from committing its transaction automatically when it finishes copying data.</p> <p>The following requirements and restrictions apply:</p> <ul style="list-style-type: none"> • This option must be the last COPY statement parameter. • You cannot combine this option with REJECTED DATA AS TABLE. • CREATE EXTERNAL TABLE AS COPY ignores this option.

Note: The RETURNREJECTED parameter is supported only for internal use by the JDBC and ODBC drivers. Be aware that the Vertica internal-use options can change without notice.

COPY Option Parser Dependencies

The following table summarizes which COPY parameters are available when loading data using the default (DELIMITER), NATIVE (binary), NATIVE VARCHAR, and FIXEDWIDTH parsers:

COPY Option	DELIMITER	NATIVE (BINARY)	NATIVE (VARCHAR)	FIXEDWIDTH
COLUMN OPTION	•	•	•	•
AUTO	•	•	•	•
DIRECT	•	•	•	•
TRICKLE	•	•	•	•
ENFORCELENGTH	•	•	•	•
EXCEPTIONS	•	•	•	•
FILLER	•	•	•	•
REJECTED DATA	•	•	•	•
ABORT ON ERROR	•	•	•	•
STREAM NAME	•	•	•	•
SKIP	•	•	•	•
SKIP BYTES				•
REJECTMAX	•	•	•	•
STDIN	•	•	•	•
UNCOMPRESSED	•	•	•	•
BZIP GZIP	•	•	•	•

COPY Option	DELIMITER	NATIVE (BINARY)	NATIVE (VARCHAR)	FIXEDWIDTH
CONCATENATED BZIP GZIP	•			•
NO COMMIT	•	•	•	•
FORMAT	•	•	•	•
NULL	•	•	•	•
DELIMITED	•			
ENCLOSED BY	•			
ESCAPE AS	•			
TRAILING NULLCOLS	•			
RECORD TERMINATOR	•			•
TRIM				•

COPY Restrictions

COPY considers the following data invalid:

- Missing columns (an input line has fewer columns than the recipient table).
- Extra columns (an input line has more columns than the recipient table).
- Empty columns for an INTEGER or DATE/TIME data type. If a column is empty for either of these types, COPY does not use the default value that was defined by the [CREATE TABLE](#) command. However, if you do not supply a column option as part of the COPY statement, the default value is used.
- Incorrect representation of a data type. For example, trying to load a non-numeric value into an INTEGER column is invalid.

When COPY encounters an empty line while loading data, the line is neither inserted nor rejected, but COPY increments the line record number. Consider this behavior when evaluating rejected records. If you return a list of rejected records and COPY encountered an empty row while loading data, the position of rejected records is incremented by one.

When loading compressed files, COPY might abort and report an error, if the file seems to be corrupted. For example, this behavior can occur if reading the header block fails.

If any primary key, unique key, or check constraints are enabled for automatic enforcement, Vertica enforces those constraints when you insert values into a table. If a violation occurs, Vertica rolls back the SQL statement and returns an error. This behavior occurs for INSERT, UPDATE, COPY, and MERGE SQL statements. Note that automatic constraint enforcement requires that you have the SELECT privilege on the table containing the constraint.

COPY Examples

The following examples show how to load data with the COPY statement using various string options.

The FORMAT, DELIMITER, NULL, and ENCLOSED BY options:

```
=> COPY public.customer_dimension (customer_since FORMAT 'YYYY')
    FROM STDIN
    DELIMITER ','
    NULL AS 'null'
    ENCLOSED BY ''';
```

The DELIMITER, NULL, and DIRECT options:

```
=> COPY a
    FROM STDIN
    DELIMITER ','
    NULL E'\\N'
    DIRECT;
```

The DELIMITER and NULL options:

```
=> COPY store.store_dimension
    FROM :input_file
    DELIMITER '|'
    NULL ''
    RECORD TERMINATOR E'\f';
```

Setting vsql Variables

The first two examples load data from STDIN. The last example uses a vsql variable (input_file). You can set a vsql variable as follows:

```
=> \set input_file ../myCopyFromLocal/large_table.zip
```

Including Multiple Source Files

COPY supports the inclusion of multiple source files in a single COPY statement.

The following example creates a table named 'sampletab.' It then copies multiple source files to the table using a single COPY statement.

```
=> CREATE TABLE sampletab (a int);
CREATE TABLE
=> COPY sampletab FROM '/home/dbadmin/one.dat', 'home/dbadmin/two.dat';
  Rows Loaded
-----
                2
(1 row)
```

You can use wildcards to indicate a group of files:

```
=> COPY myTable FROM 'hdfs:///mydirectory/ofmanyfiles/*.dat';
```

Wildcards can include regular expressions:

```
=> COPY myTable FROM 'hdfs:///mydirectory/*_[0-9]';
```

Distributing a Load

The following example shows how you can load data that is shared across all nodes. Vertica distributes the load across all nodes, if possible.

```
=> COPY sampletab FROM '/data/file.dat' ON ANY NODE;
```

This example shows how to load data from two files. Because the first load file does not specify nodes (or ON ANY NODE), the initiator performs the load. Loading the second file is distributed across all nodes.

```
=> COPY sampletab FROM '/data/file1.dat', '/data/file2.dat' ON ANY NODE;
```

This example shows how to specify different nodes for each load file. Vertica distributes the load of file1.dat across v_vmart_node0001 and v_vmart_node0002 and distributes the load of file2.dat across v_vmart_node0003 and v_vmart_node0004.

```
=> COPY sampletab FROM '/data/file1.dat' ON (v_vmart_node0001, v_vmart_node0002),
  '/data/file2.dat' ON (v_vmart_node0003, v_vmart_node0004);
```

ON ANY NODE is the default for loads from HDFS. You do not need to specify it.

Loading Data from HDFS

This example shows how you can load a file stored in HDFS. ON ANY NODE is not required.

```
=> COPY t FROM 'hdfs:///opt/data/file1.dat';
```

This example shows how you can load data from a particular HDFS name service (testNS). You specify a name service if your database is configured to read from more than one HDFS cluster.

```
=> COPY t FROM 'hdfs://testNS/opt/data/file2.csv';
```

Loading Data into a Flex Table

This statement creates a Flex table, and copies JSON data into the table, using the flex table parser, `fjsonparser`:

```
=> CREATE FLEX TABLE darkdata();  
CREATE TABLE  
=> COPY tweets FROM '/myTest/Flexible/DATA/tweets_12.json' parser fjsonparser();  
Rows Loaded  
-----  
12  
(1 row)
```

Using Named Pipes

COPY supports named pipes that follow the same naming conventions as file names on the given file system. Permissions are `open`, `write`, and `close`.

This statement creates the named pipe, `pipe1`, and sets two vsql variables, `dir` and `file`:

```
=> \! mkfifo pipe1  
=> \set dir `pwd`/  
=> \set file '':dir'pipe1''
```

This statement copies an uncompressed file from the named pipe:

```
=> \! cat pf1.dat > pipe1 &  
=> COPY large_tbl FROM :file delimiter '|';  
=> SELECT * FROM large_tbl;  
=> COMMIT;
```

Loading Compressed Data

This statement copies a GZIP file from a named pipe and uncompresses it:

```
=> \! gzip pf1.dat
=> \! cat pf1.dat.gz > pipe1 &
=> COPY large_tbl FROM :file ON site01 GZIP delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
=> \!gunzip pf1.dat.gz
```

This statement copies a BZIP file from a named pipe and then uncompresses it:

```
=> \! bzip2 pf1.dat
=> \! cat pf1.dat.bz2 > pipe1 &
=> COPY large_tbl FROM :file ON site01 BZIP delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
=> \! bunzip2 pf1.dat.bz2
```

This statement copies an LZO file from a named pipe and then uncompresses it:

```
=> \! lzop pf1.dat
=> \! cat pf1.dat.lzo > pipe1 &
=> COPY large_tbl FROM :file ON site01 LZO delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
=> \! lzop -d pf1.dat.lzo
```

See Also

- [SQL Data Types](#)
- [ANALYZE_CONSTRAINTS](#)
- [Choosing a Load Method](#) in the Administrator's Guide
- [CREATE EXTERNAL TABLE AS COPY](#)
- [Directory.GetFiles Method](#)
- [Bulk-Loading Data](#) in the Administrator's Guide
- [Loading Fixed-Width Format Data](#) in the Administrator's Guide
- [Loading Binary \(Native\) Data](#) in the Administrator's Guide

- [Bulk Loading Data into Flex Tables](#) in the Administrator's Guide
- [Manipulating Source Data Columns](#) in the Administrator's Guide
- [Linux Manual Page for Glob \(7\)](#)
- [Tracking Load Exceptions and Rejections Status](#) in the Administrator's Guide
- [Transforming Data During Loads](#) in the Administrator's Guide

COPY LOCAL

Using the COPY statement with its LOCAL option lets you load a data file on a client system, rather than on a cluster host. COPY LOCAL supports the STDIN and 'pathToData' parameters, but not the [ON nodename] clause. COPY LOCAL does not support multiple file batches in NATIVE or NATIVE VARCHAR formats. COPY LOCAL does not support reading ORC or Parquet files; use ON NODE instead.

The COPY LOCAL option is platform independent. The statement works in the same way across all supported Vertica platforms and drivers. For more details about using COPY LOCAL with supported drivers, see the Connecting to Vertica section for your platform.

Note: On Windows clients, the path you supply for the COPY LOCAL file is limited to 216 characters due to limitations in the Windows API.

Invoking COPY LOCAL does not automatically create exceptions and rejections files, even if exceptions occur. You cannot save exceptions and rejections to a table with the rejected data as table parameter. For information about saving such files, see [Capturing Load Rejections and Exceptions](#) in the Administrator's Guide.

Privileges

User must have INSERT privilege on the table and USAGE privilege on the schema.

How Copy Local Works

COPY LOCAL loads data in a platform-neutral way. The COPY LOCAL statement loads all files from a local client system to the Vertica host, where the server processes the files. You can copy files in various formats: uncompressed, compressed, fixed-width format, in bzip or gzip

format, or specified as a bash glob. Files of a single format (such as all bzip, or gzip) can be comma-separated in the list of input files. You can also use any of the applicable COPY statement options (as long as the data format supports the option). For instance, you can define a specific delimiter character, or how to handle NULLs, and so forth.

Note: The Linux `glob` command returns files that match the pattern you enter, as specified in the [Linux Manual Page for Glob \(7\)](#). For ADO.net platforms, specify patterns and wildcards as described in the .NET [Directory.GetFiles Method](#).

For more information about using the COPY LOCAL option to load data, see [COPY](#) for syntactical descriptions, and [Using COPY and COPY LOCAL](#) for detailed examples.

The Vertica host uncompresses and processes the files as necessary, regardless of file format or the client platform from which you load the files. Once the server has the copied files, Vertica maintains performance by distributing file parsing tasks, such as encoding, compressing, uncompressing, across nodes.

Viewing Copy Local Operations in a Query Plan

When you use the COPY LOCAL option, the GraphViz query plan includes a label for Load-Client-File, rather than Load-File. Following is a section from a sample query plan:

```
-----  
PLAN:  BASE BULKLOAD PLAN  (GraphViz Format)  
-----  
digraph G {  
graph [rankdir=BT, label = " BASE BULKLOAD PLAN \nAll Nodes Vector:  
\n\n node[0]=initiator (initiator) Up\n", labelloc=t, labeljust=l ordering=out]  
.  
.  
.  
10[label = "Load-Client-File(/tmp/diff) \nOutBlk=[UncTuple]",  
color = "green", shape = "ellipse"];
```

COPY FROM VERTICA

Imports data from another Vertica database. COPY FROM VERTICA is similar to [COPY](#), but accepts only a subset of its parameters.

Important: The source database can be one major release behind the target database.

Syntax

```
COPY [schema.]target-table
... [( target-column[,...])]
... FROM VERTICA database [schema.]source-table
... [(source-column[,...])]
... [Load-method]
... [STREAM NAME 'stream name']
... [NO COMMIT]
```

Parameters

<p>[<i>schema</i> .] <i>target-table</i></p>	<p>The table in the local database to store the copied data.</p>
<p><i>target-column</i></p>	<p>A target table column to store the copied data. If you specify target columns, COPY FROM VERTICA writes only to those columns. If you omit specifying target columns, Vertica writes to target table columns as described below, in Source and Target Column Mapping.</p> <p>Note: You cannot use column fillers as part of the column definition.</p>
<p><i>database</i></p>	<p>The source database of the data to copy. A connection to this database must already exist in the current session.</p>
<p>[<i>schema</i> .] <i>source-table</i></p>	<p>The table that is the source of the copied data.</p>
<p><i>source-column</i></p>	<p>A source table column to copy. If you specify source columns, only these columns are copied from the source table. If you omit specifying source columns, Vertica copies columns from the source table, as described below in Source and Target Column Mapping.</p>
<p><i>Load-method</i></p>	<p>Specifies how to load data into the database, one of the following:</p> <ul style="list-style-type: none"> AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads.

	<ul style="list-style-type: none"> • DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads. • TRICKLE: Loads data only into WOS, suitable for frequent incremental loads. <p>This option is invalid for external tables.</p> <p>For details, see Choosing a Load Method in the Administrator's Guide.</p>
STREAM NAME	<p>Specifies a COPY load stream identifier. Using a stream name helps to quickly identify a particular load. The STREAM NAME value that you specify in the load statement appears in the stream column of the LOAD_STREAMS system table.</p> <p>By default, Vertica names streams by table and file name. For example, if you have two files (f1, f2) in Table A, their stream names are A-f1, A-f2, respectively.</p> <p>To name a stream:</p> <pre>=> COPY mytable FROM myfile DELIMITER ' ' DIRECT STREAM NAME 'My stream name';</pre>
NO COMMIT	<p>Prevents COPY from committing its transaction automatically when it finishes copying data. For details, see Overriding COPY Auto Commit in the Administrator's Guide.</p>

Privileges

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema

Connecting to the Source Database

Before you can import data from another database, you must establish a connection to the source database with [CONNECT](#). See [Copying Data from Another Vertica Database](#) for details.

By default, `COPY FROM VERTICA` copies or imports data over the Vertica private network. Connecting to a public network requires some configuration. For information about using this statement to copy data across a public network, see [Using Public and Private IP Networks](#).

The copy operation fails if either side of the connection is a single-node cluster installed to `localhost`, or you do not specify a host name or IP address.

Source and Target Column Mapping

You can optionally name a subset of source and target columns to participate in the copy operation. `COPY FROM VERTICA` attempts to match columns in the source table with corresponding columns in the destination table.

The following table compares the different combinations of naming source and target columns, and the requirements that pertain to each option.

	Omit source columns	Specify source columns
Omit target columns	<p>Match all columns in source table to columns in target table.</p> <p>The number of columns in the two tables can differ, but the target table must have at least as many columns as the source table.</p>	<p>Match named source table columns to target table columns.</p> <p>The number of columns in the two tables can differ, but the target table must have at least as many columns as the number of specified source columns.</p>
Specify target columns	<p>Match source columns to the named target columns.</p> <p>The number of named target columns must equal the number of columns in the source table.</p>	<p>Match named source columns to named target columns.</p> <p>The number of source and target columns must be equal.</p>

Node Failure During COPY

See [Handling Node Failure During Copy/Export](#) in the Administrator's Guide.

Examples

See [Copying Data from Another Vertica Database](#) in the Administrator's Guide.

See Also

[EXPORT TO VERTICA](#)

CREATE ACCESS POLICY

Creates a secure access policy to prevent unauthorized users from accessing potentially sensitive information. You can create access policies for table rows and columns. Access policies are a technique of on-the-fly query modification in which the query excludes rows or modifies data from a column in the results returned to the user running the query. Access policies allow different users to run the same query and receive different results.

Column access policies limit access to specific column in a table. Creating a column access policy depends on the expressions specified when creating the policy. The expression is substituted for the column's actual value in any data fetched from the table.

Row access policies limits access to a specific row in a table. You must use a WHERE clause to set the access policy's condition. Only rows that satisfy the WHERE clause are fetched from the table.

For information on how implementing access policies affects how you manage data, see [Working With Access Policies](#).

Syntax

```
CREATE ACCESS POLICY ON [schema.]table-name  
... { FOR COLUMN column-name | FOR ROWS WHERE } expression ENABLE
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example:
---------------	---

	<code>myschema.thisDBObject</code>
<i>table-name</i>	The table that contains the target column.
<i>column-name</i>	The column on which to apply an access policy.
<i>expression</i>	<p>An SQL expression that specifies conditions for column or row access:</p> <ul style="list-style-type: none"> • In a column access policy, the expression defines what value is returned when this column is fetched. The expression can contain conditions such as a role to determine what value to return. The expression might return null if no data is returned for this column. • In a row access policy, only rows for which the expression is true are returned. <p>The expression in the example below shows the first 5 characters of the key in the <code>customer_key</code> column:</p> <pre>=> CREATE ACCESS POLICY ON customer_dimension FOR COLUMN customer_key substr(customer_key, 1, 5) ENABLE;</pre>
ENABLE	Enables the access policy. Always add this to end of the statement when creating an access policy.

Privileges

One of the following:

- dbadmin
- Superuser

Examples

Create access policy per role on column

Add an expression to the policy to specify the access each role receives. A manager can access the complete customer number, while an operator can only see a portion of the number:

```
=> CREATE ACCESS POLICY on customer FOR column customer_number
CASE
```

```
WHEN enabled_role ('manager') then customer_number  
WHEN enabled_role ('operator') then substr (customer_number, 8,2)  
ELSE NULL  
END  
ENABLE;
```

Create access policy for rows

```
=> CREATE ACCESS POLICY ON customer FOR ROWS WHERE cid1>1 ENABLE;
```

See also:

[Access Policies](#)

[Working With Access Policies](#)

[Column Access Policy](#)

[Row Access Policy](#)

CREATE AUTHENTICATION

Creates and enables an authentication method associated with users or roles. When you create an authentication method using CREATE AUTHENTICATION, Vertica enables it automatically.

Syntax

```
CREATE AUTHENTICATION auth-method-name  
  METHOD auth-type  
  { LOCAL | HOST [ { TLS | NO TLS } ] host-ip-address }
```

Parameters

Name	Data Type	Description
<i>auth-method-name</i>	VARCHAR	Name of the authentication method you want to create, where <i>auth-method-</i>

Name	Data Type	Description
		<i>name</i> conforms to conventions described in Identifiers .
<i>auth-type</i>	VARCHAR	<p>Name of the authentication method you want to use.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • 'gss' • 'ident' • 'ldap' • 'hash' • 'reject' • 'trust' • 'tls'
{ LOCAL HOST [{ TLS NO TLS }] }	N/A	<p>The access method the client uses to connect.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • LOCAL—Matches connection attempts made using local domain sockets. When using the local connection type, do not specify the <i><address></i> parameter. • HOST—Matches connection attempts made using TCP/IP. Vertica attempts connection using a plain (non-SSL/TLS) or SSL/TLS-wrapped TCP socket. • HOST TLS—Matches an SSL/TLS TCP connection only. • HOST NO TLS—Matches a plain TCP socket only.

Name	Data Type	Description
<i>host-ip-address</i>	VARCHAR	Required if you specify HOST. Vertica supports IPv4 and IPv6 addresses. For more information, see IPv4 and IPv6 for Client Authentication .

Privileges

Must have DBADMIN privileges.

Examples

This example shows you how to create an authentication method named `localpwd` to authenticate users who are trying to log in from a local host using a password:

```
=> CREATE AUTHENTICATION localpwd METHOD 'hash' LOCAL;
```

This example shows you how to create an authentication method named `v_ldap` that uses LDAP over TLS to authenticate users logging in from the host with the IPv4 address 10.0.0.0/23:

```
=> CREATE AUTHENTICATION v_ldap METHOD 'ldap' HOST TLS '10.0.0.0/23';
```

This example shows you how to create an authentication method named `v_kerberos` to authenticate users that are trying to connect from any host in the networks 2001:0db8:0001:12xx:

```
=> CREATE AUTHENTICATION v_kerberos METHOD 'gss' HOST '2001:db8:1::1200/56';
```

This example shows you how to create an authentication method named, `RejectNoSSL`, that rejects users from any IP address that are trying to authenticate without SSL/TLS:

```
=> CREATE AUTHENTICATION RejectNoSSL_IPv4 METHOD 'reject' HOST NO TLS '0.0.0.0/0'; --IPv4  
=> CREATE AUTHENTICATION RejectNoSSL_IPv6 METHOD 'reject' HOST NO TLS ':::0'; --IPv6
```

See Also

- [ALTER AUTHENTICATION](#)
- [DROP AUTHENTICATION](#)
- [GRANT \(Authentication\)](#)
- [REVOKE \(Authentication\)](#)
- [IPv4 and IPv6 for Client Authentication](#)

CREATE EXTERNAL TABLE AS COPY

The CREATE EXTERNAL TABLE AS COPY statement creates a query definition for a table external to your Vertica database. This statement is a combination of the [CREATE TABLE](#) and [COPY](#) statements, supporting a subset of each statement's parameters, noted below. You can also use user-defined load extension functions (UDLs) to create external tables. For more information about UDL syntax, see [User Defined Load \(UDL\)](#).

You might discover that you need to change a column data type. For example, a VARCHAR might have longer data values than you anticipated at table-creation time. You can use [ALTER TABLE](#) to change the data types of columns instead of dropping and recreating the table.

You can use CREATE EXTERNAL TABLE AS COPY with int, long varchar, and long varbinary data types.

Note: Vertica does not create a superprojection for an external table, since they are not stored in the database. External tables consist of query definitions, so their data is available only at query time.

Privileges

You must be a database superuser to create external tables, unless you have created a user-accessible storage location to which the COPY refers, see [CREATE LOCATION](#). If external tables exist, you must also be a database superuser to access them through a select statement.

You must have full access (including [SELECT](#)) to an external table that a user has privileges to create. The database superuser must also grant READ access to the USER-accessible storage location, see [GRANT \(Storage Location\)](#).

Syntax

```
CREATE EXTERNAL TABLE [ IF NOT EXISTS ] [schema.]table-name
... ( Column-Definition [ , ... ] )
[ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
... AS COPY
... [ ( { column-as-expression | column }
..... [ DELIMITER [ AS ] 'char' ]
..... [ ENCLOSED [ BY ] 'char' ]
..... [ ENFORCELENGTH ]
..... [ ESCAPE [ AS ] 'char' | NO ESCAPE ]
..... [ FILLER datatype ]
..... [ FORMAT 'format' ]
..... [ NULL [ AS ] 'string' ]
..... [ TRIM 'byte' ]
... [, ... ] ) ]
... [ COLUMN OPTION ( column
..... [ DELIMITER [ AS ] 'char' ]
..... [ ENCLOSED [ BY ] 'char' ]
..... [ ENFORCELENGTH ]
..... [ ESCAPE [ AS ] 'char' | NO ESCAPE ]
..... [ FORMAT 'format' ]
..... [ NULL [ AS ] 'string' ]
..... [ TRIM 'byte' ]
... [, ... ] ) ]
[ FROM
... 'pathToData' [ ON nodename | ON ANY NODE | ON (nodeset) ]
..... [ BZIP | GZIP | LZO | UNCOMPRESSED ] [, ...]
...[ NATIVE
.....| FIXEDWIDTH COLSIZES {( integer ) [,....]}
.....| NATIVE VARCHAR
.....| ORC
.....| PARQUET
...[ WITH ] [ SOURCE source([arg=value [,...]]) ]
...]
...[ ABORT ON ERROR ]
...[ DELIMITER [ AS ] 'char' ]
...[ ENCLOSED BY 'char' [ AND 'char' ] ]
...[ ENFORCELENGTH ]
...[ ERROR TOLERANCE ]
...[ ESCAPE AS 'char' | NO ESCAPE ]
...[ EXCEPTIONS 'path' [ ON nodename ] [, ...] ]
...[ WITH ] ...[ FILTER filter([arg=value [,...]]) ] [...]
...[ NULL [ AS ] 'string' ]
...[ WITH ] [ PARSER parser([arg=value [,...]]) ]
...[ RECORD TERMINATOR 'string' ]
...[ REJECTED DATA 'path' [ ON nodename ] [, ...] ]
...[ REJECTMAX integer ]
...[ SKIP integer ]
...[ SKIP BYTES integer ]
...[ TRAILING NULLCOLS ]
...[ TRIM 'byte' ]
```

Parameters

For all supported parameters, see the [CREATE TABLE](#) and [COPY](#) statements. For information on using this statement with UDLs see [Load \(UDLs\)](#).

Examples

Examples of external table definitions:

```
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat' DELIMITER ',';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM 'hdfs:///dat/ext1.dat';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat.bz2' BZIP DELIMITER ',';
=> CREATE EXTERNAL TABLE ext1 (x integer, y integer) AS COPY (x as '5', y) FROM '/tmp/ext1.dat.bz2'
BZIP DELIMITER ',';
```

To allow users without superuser access to use these tables, create a location for 'user' usage and grant access to it. This example shows granting access to a user named Bob to any external table whose data is located under /tmp (including in subdirectories to any depth):

```
=> CREATE LOCATION '/tmp' ALL NODES USAGE 'user';
=> GRANT ALL ON LOCATION '/tmp' to Bob;
```

See Also

- [Physical Schema](#)
- [CREATE TABLE](#)
- [CREATE FLEX TABLE](#)
- [SELECT](#)
- [Using External Tables](#)

CREATE FAULT GROUP

Creates a fault group, which can contain the following:

- One or more nodes
- One or more child fault groups
- One or more nodes and one or more child fault groups

The `CREATE FAULT GROUP` statement creates an empty fault group. You must run the [ALTER FAULT GROUP](#) statement to add nodes or other fault groups to an existing fault group.

Syntax

```
CREATE FAULT GROUP name
```

Parameters

<i>name</i>	Specifies the name of the fault group to create, where <i>name</i> conforms to conventions described in Identifiers . You must provide distinct names for each fault group you create.
-------------	--

Privileges

Must be a superuser to create a fault group.

Example

The following command creates a fault group called `parent0`:

```
=> CREATE FAULT GROUP parent0;  
CREATE FAULT GROUP
```

To add nodes or other fault groups to the `parent0` fault group, run the [ALTER FAULT GROUP](#) statement.

See Also

- [ALTER FAULT GROUP](#)
- [V_CATALOG.FAULT_GROUPS](#)
- [V_CATALOG.CLUSTER_LAYOUT](#)
- [Fault Groups](#)
- [High Availability With Fault Groups](#)

CREATE FLEX TABLE

Creates a flex table in the logical schema. Declaring columns (or other supported parameters) is optional. If you do not declare any column definitions, the statement creates two columns automatically:

`__raw__` : A LONG VARBINARY type column to store any unstructured data you load. This column has a NOT NULL constraint by default. .

`__identity__` : An IDENTITY column. Flex tables use this value for segmentation and sorting, when no other column definition exists.

Additionally, creating any flex table results in three associated objects:

- A flex table (*flex_table*) named in this statement
- A related keys table, called *flex_table_keys*
- A related view, called *flex_table_view*

Both the flex table and its associated `_keys` table are required to use flex tables successfully. The `_keys` table and `_view` are subservient objects of the flex table. Neither can exist without the flex table.

For more details about creating and using flex tables, see [Creating Flex Tables](#) and other sections in Using Flex Tables.

CREATE FLEX TABLE supports many of the parameters available when creating columnar tables, but not all. This section presents the optional use of column definitions, and the subset of supported parameters.

You can also create flex external tables, with some syntactical variations, as described in [CREATE FLEX EXTERNAL TABLE AS COPY](#) .

You cannot partition a flex table on any virtual column (key).

Note: Vertica does not support flexible global temporary tables.

Syntax

```
CREATE {FLEX | FLEXIBLE} TABLE [ IF NOT EXISTS ] [schema.]table-name {  
... ( [ column-definition [ , ... ] ] )  
... | [ table-constraint ( column_name, ... ) ]  
... | [ column-name-list (create table) ]  
}  
  
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]  
... [ ORDER BY table-column [ , ... ] ]  
... [ ENCODED BY column-definition [ , ... ] ]  
... [ hash-segmentation-clause  
..... | UNSEGMENTED { NODE node | ALL NODES } ]  
... [ KSAFE [k_num] ]  
... [ PARTITION BY partition-expression ]  
... [ AS SELECT (column-name-list) FROM (table-name) ]
```

Parameters

See the [CREATE TABLE](#) statement for all parameter descriptions.

Unsupported CREATE Flex Table Option

You cannot use the following options when creating a flex table:

```
... AS [COPY] [ [ AT EPOCH LATEST ] ... | [ AT TIME 'timestamp' ] ]  
.....[ /*+ direct */ ] query  
... | [ LIKE [schema.]existing-table [ INCLUDING PROJECTIONS | EXCLUDING PROJECTIONS ] ]
```

Default Flex Table and Keys Table Projections

Vertica automatically creates superprojections for both the flex table and keys tables when you create them.

If you create a flex table with one or more of the ORDER BY, ENCODED BY, SEGMENTED BY, or KSAFE clauses, the clause information is used to create projections. If no clauses are in use, Vertica uses the following defaults for unspecified aspects:

Table	order_by	encoded_by	Segmentation	Ksafe
flexible table	__identity__	none	by hash __identity__	1
keys_table	frequency	none	replicated/unsegmented all nodes	1

Note: When you build a view for a flex table (see [BUILD_FLEXTABLE_VIEW](#)), the view is ordered by frequency, desc, and key_name.

Privileges

To create a flex table, you must have CREATE privileges on the table schema.

Examples

The following example creates a flex table named darkdata without specifying any column information. Vertica creates a default superprojection and buddy projection as part of creating the table:

```
=> CREATE FLEXIBLE TABLE darkdata();  
CREATE TABLE
```

The following example creates a table called darkdata1 with one column definition (date_col). The statement specifies the partition by clause to partition the data by year. Vertica creates a default superprojection and buddy projections as part of creating the table:

```
=> CREATE FLEX TABLE darkdata1 (date_col date NOT NULL) partition by  
  extract('year' from date_col);  
CREATE TABLE
```

See Also

- [Physical Schema](#)
- [COPY](#)
- [CREATE EXTERNAL TABLE AS COPY](#)
- [CREATE FLEX EXTERNAL TABLE AS COPY](#)
- [CREATE TABLE](#)

- [PARTITION_PROJECTION](#)
- [PARTITION_TABLE](#)
- [SELECT](#)
- [Using Table Partitions](#)
- [Using External Tables](#)

CREATE FLEX EXTERNAL TABLE AS COPY

The CREATE FLEX EXTERNAL TABLE AS COPY statement creates a flexible external table. This statement is a combination of the [CREATE FLEX TABLE](#) and [COPY](#) statements, supporting a subset of each statement's parameters, as noted below. You can also use user-defined load extension functions (UDLs) to create external flex tables. For more information about UDL syntax, see [User Defined Load \(UDL\)](#) and [COPY](#). For more details about creating and using flex tables, see [Using Flex Tables](#).

Note: Vertica does not create a superprojection for an external table when you create it.

Privileges

Must be a database superuser to create external tables, unless the superuser has created a user-accessible storage location to which the COPY refers, as described in [CREATE LOCATION](#). If external tables exist, you must also be a database superuser to access them through a select statement.

Permission requirements for flex external tables differ from other flex tables. You must have full access (including SELECT) to an external table that a user has privileges to create. The database superuser must also grant READ access to the USER-accessible storage location, see [GRANT \(Storage Location\)](#).

For more details about creating and using flex tables, see [Creating Flex Tables](#) and other sections in the [Using Flex Tables](#).

Syntax

```
CREATE {FLEX | FLEXIBLE} EXTERNAL TABLE [ IF NOT EXISTS ] [schema.]table-name {  
... ( [ Column-Definition [ , ... ] ] )
```

```

} [INCLUDE | EXCLUDE [SCHEMA] PRIVILEGES]
... AS COPY .... [ ( { column-as-expression | column } .....[ FILLER datatype ] ]
[ FROM
... 'pathToData' [ ON nodename | ON ANY NODE | ON (nodeset) ]
..... [ BZIP | GZIP | LZO | UNCOMPRESSED ] [, ...]
...[ WITH ]
...[ SOURCE source(arg='value')]
...[ FILTER filter(arg='value') ]
...[ PARSER flexparser(arg='value') ]
...[ DELIMITER [ AS ] 'char' ]
...[ TRAILING NULLCOLS ]
...[ NULL [ AS ] 'string' ]
...[ ESCAPE [ AS ] 'char' | NO ESCAPE ]
...[ ENCLOSED [ BY ] 'char' ]
...[ RECORD TERMINATOR 'string' ]
...[ SKIP integer ]
...[ SKIP BYTES integer ]
...[ TRIM 'byte' ]
...[ REJECTMAX integer ]
...[ EXCEPTIONS 'path' [ ON nodename ] [, ...] ]
...[ REJECTED DATA 'path' [ ON nodename ] [, ...] ]
...[ ENFORCELENGTH ]
...[ ABORT ON ERROR ]

```

Parameters

The following parameters from the parent statements are not supported in the CREATE FLEXIBLE EXTERNAL TABLE AS COPY statement:

CREATE TABLE
AS AT EPOCH LAST
AT TIME 'timestamp'
ORDER BY table-column [,...]
ENCODED BY
hash-segmentation-clause
UNSEGMENTED {node node all}
KSAFE [k_num]
PARTITION BY partition-clause

COPY
FROM STDIN
FROM LOCAL
DIRECT

COPY
TRICKLE
NO COMMIT

For all supported parameters, see the [CREATE TABLE](#) and [COPY](#) statements.

Notes

Canceling a CREATE FLEX EXTERNAL TABLE AS COPY statement can cause unpredictable results. Vertica recommends that you allow the statement to finish, then use [DROP TABLE](#) once the table exists.

Examples

To create an external flex table:

```
=> CREATE flex external table mountains() AS COPY FROM 'home/release/KData/kmm_ountains.json' PARSER
fjsonparser();
CREATE TABLE
```

As with other flex tables, creating an external flex table produces two regular tables: the named table and its associated `_keys` table. The keys table is not an external table:

```
=> \dt mountains
          List of tables
 Schema | Name      | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | mountains | table | release |
(1 row)
```

You can use the helper function, [COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW](#), to compute keys and create a view for the external table:

```
=> SELECT compute_flextable_keys_and_build_view ('appLog');

          compute_flextable_keys_and_build_view
-----
Please see public.appLog_keys for updated keys
The view public.appLog_view is ready for querying
(1 row)
```

1. Check the keys from the `_keys` table for the results of running the helper application:

```
=> SELECT * FROM appLog_keys;
      key_name          | frequency | data_type_guess
-----+-----+-----
contributors          |         8 | varchar(20)
coordinates            |         8 | varchar(20)
created_at            |         8 | varchar(60)
entities.hashtags     |         8 | long varbinary(186)
.
.
.
retweeted_status.user.time_zone |         1 | varchar(20)
retweeted_status.user.url   |         1 | varchar(68)
retweeted_status.user.utc_offset |         1 | varchar(20)
retweeted_status.user.verified |         1 | varchar(20)
(125 rows)
```

2. Query from the external flex table view:

```
=> SELECT "user.lang" FROM appLog_view;
 user.lang
-----
it
en
es
en
en
es
tr
en
(12 rows)
```

See Also

- [COPY Parameters](#)
- [CREATE EXTERNAL TABLE AS COPY](#)
- [CREATE TABLE](#)
- [CREATE FLEX TABLE](#)
- [SELECT](#)

CREATE FUNCTION Statements

CREATE FUNCTION statements can create two different kinds of functions:

- User defined SQL functions let you define and store commonly-used SQL expressions as a function. User defined SQL functions are useful for executing complex queries and combining Vertica built-in functions. You simply call the function name you assigned in your query.
- User defined scalar functions (UDSFs) take in a single row of data and return a single value. These functions can be used anywhere a native Vertica function or statement can be used, except CREATE TABLE with its PARTITION BY or any segmentation clause.

While you use CREATE FUNCTION to create both SQL and scalar functions, you use a different syntax for each function type. For more information, see:

- [CREATE FUNCTION \(SQL Functions\)](#)
- [CREATE FUNCTION \(UDF\)](#)

About Creating User Defined Transform Functions (UDTFs)

You can use a similar SQL statement to create user-defined transform functions. User defined transform functions (UDTFs) operate on table segments and return zero or more rows of data. The data they return can be an entirely new table, unrelated to the schema of the input table, including having its own ordering and segmentation expressions. They can only be used in a query's SELECT list. For details about creating a UDTF, see [CREATE TRANSFORM FUNCTION](#).

CREATE AGGREGATE FUNCTION

Adds a user-defined aggregate function (UDAF) stored in a shared Linux library to the catalog. You must have already loaded this library using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes data values to the code in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] AGGREGATE FUNCTION [schema.]function-name
... AS LANGUAGE 'Language' NAME 'factory' LIBRARY Library-name;
```

Parameters

OR REPLACE	If you do not supply this parameter, CREATE AGGREGATE FUNCTION fails if an existing function matches the name and parameters of the function you are trying to define. If you do supply this parameter, the new function definition overwrites the old.
<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>function-name</i>	The name of the function to create. If the function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.
LANGUAGE ' <i>Language</i> '	The language used to develop this function, currently C++ only. Default value: C++
NAME ' <i>factory</i> '	The name of the factory class in the shared library that generates the object to handle the function's processing.
LIBRARY <i>Library-name</i>	The name of the shared library that contains the C++ object to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.

Notes

- The parameters and return value for the function are automatically determined by the CREATE AGGREGATE FUNCTION statement, based on data supplied by the factory class.

- When a User Defined Aggregate function that is defined multiple times with arguments of different data types is called, Vertica selects the function whose input parameters match the parameters in the function call to perform the processing.
- You can return a list of all SQL functions and User Defined Functions (including aggregates) by querying the system table `V_CATALOG.USER_FUNCTIONS` or executing the vsql meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

Privileges

- Only a superuser can create or drop a User Defined Aggregate library.
- To create a User Defined Aggregate function, the user must have CREATE and USAGE privileges on the schema and USAGE privileges on the library.
- To use a User Defined Aggregate, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined function. See [GRANT \(User Defined Extension\)](#) and [REVOKE \(User Defined Extension\)](#).

Examples

The following example demonstrates loading a library named `AggregateFunctions` then defining a function named `ag_avg` and `ag_cat` that are mapped to the `ag_cat` `AverageFactory` and `ConcatenateFactory` classes in the library:

```
=> CREATE LIBRARY AggregateFunctions AS '/opt/vertica/sdk/examples/build/AggregateFunctions.so';
CREATE LIBRARY
=> CREATE AGGREGATE FUNCTION ag_avg AS LANGUAGE 'C++' NAME 'AverageFactory'
    library AggregateFunctions;
CREATE AGGREGATE FUNCTION
=> CREATE AGGREGATE FUNCTION ag_cat AS LANGUAGE 'C++' NAME 'ConcatenateFactory'
    library AggregateFunctions;
CREATE AGGREGATE FUNCTION
=> \x
Expanded display is on.
select * from user_functions;
-[ RECORD 1 ]-----+-----
schema_name      | public
function_name    | ag_avg
procedure_type   | User Defined Aggregate
function_return_type | Numeric
function_argument_type | Numeric
function_definition | Class 'AverageFactory' in Library 'public.AggregateFunctions'
volatility       |
is_strict        | f
is_fenced        | f
```

```
comment          |
-[ RECORD 2 ]-----+-----
schema_name      | public
function_name    | ag_cat
procedure_type   | User Defined Aggregate
function_return_type | Varchar
function_argument_type | Varchar
function_definition | Class 'ConcatenateFactory' in Library 'public.AggregateFunctions'
volatility       |
is_strict        | f
is_fenced        | f
comment          |
```

See Also

- [CREATE LIBRARY](#)
- [DROP AGGREGATE FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Developing User-Defined Extensions \(UDxs\)](#)
- [Aggregate Functions \(UDAFs\)](#)

CREATE ANALYTIC FUNCTION

Associates a User Defined Analytic Function (UDAnF) stored in a shared Linux library with a SQL function name. You must have already loaded the library containing the UDAnF using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes the arguments to the analytic function in the library to process.

Syntax

```
CREATE [ OR REPLACE ] ANALYTIC FUNCTION function-name
... AS [ LANGUAGE 'Language' ] NAME 'factory'
... LIBRARY library_name
... [ FENCED | NOT FENCED ];
```

Parameters

<i>function-name</i>	The name to assign to the UDAFn. This is the name you use in your SQL statements to call the function.
LANGUAGE ' <i>Language</i> '	The language used to develop this function, one of the following: <ul style="list-style-type: none">• C++• Java Default Value: C++
NAME ' <i>factory</i> '	The name of the C++ factory class in the shared library that generates the object to handle the function's processing.
LIBRARY <i>library-name</i>	The name of the shared library that contains the C++ object to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.
FENCED NOT FENCED	Enables or disables Fenced Mode for this function. Default Value: FENCED

Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.
- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.
- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

Usage Considerations

- The parameters and return value for the function are automatically determined by the CREATE ANALYTIC FUNCTION statement, based on data supplied by the factory class.
- You can assign multiple functions the same name if they accept different sets of arguments. See [Overloading Your UDX](#) in Extending Vertica for more information.
- You can return a list of all UDFs by querying the system table [V_CATALOG.USER_FUNCTIONS](#). Users see only the functions on which they have EXECUTE privileges.

Examples

This example shows how to create an analytic function named `an_rank` based on the factory class named `RankFactory` in the `AnalyticFunctions` library..

```
=> CREATE ANALYTIC FUNCTION an_rank AS LANGUAGE 'C++'  
    NAME 'RankFactory' LIBRARY AnalyticFunctions;
```

See Also

[Analytic Functions \(UDAnFs\)](#)

CREATE FILTER

Adds a user-defined load filter function. You must have already loaded this library using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] FILTER [schema.]function-name  
... AS LANGUAGE 'Language' NAME 'factory' LIBRARY 'Library-name'  
... [ FENCED | NOT FENCED ];
```

Parameters

OR REPLACE	If you do not supply this parameter, the CREATE FILTER statement fails if an existing function matches the name and parameters of the filter function you are trying to define. If you do supply this parameter, the new filter function definition overwrites the old.
<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>function-name</i>	The name of the filter function to create. If the filter function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.
LANGUAGE ' <i>Language</i> '	<p>The language used to develop this function, one of the following:</p> <ul style="list-style-type: none"> • C++ • Java
NAME ' <i>factory</i> '	<p>The name of the factory class in the shared library that generates the object to handle the filter function's processing. This is the same name used by the RegisterFactory class.</p>
LIBRARY <i>Library-name</i>	The name of the shared library that contains the C++ object to perform the processing for this filter function. This library must have been previously loaded using the CREATE LIBRARY statement.
FENCED NOT FENCED	Enables or disables Fenced Mode for this function. Fenced mode is enabled by default.

Notes

- The parameters and return value for the filter function are automatically determined by the CREATE FILTER statement, based on data supplied by the factory class.
- You can return a list of all SQL functions and User Defined Functions by querying the system table `V_CATALOG.USER_FUNCTIONS` or executing the vsql meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

Privileges

- Only a superuser can create or drop a function that uses a UDx library.
- To use a User Defined Filter, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined filter function. See [GRANT \(User Defined Extension\)](#) and [REVOKE \(User Defined Extension\)](#).

Important: Installing an untrusted UDL function can compromise the security of the server. UDx's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to UDx's to untrusted users.

Example

The following example demonstrates loading a library named `iConverterLib`, then defining a function named `Iconverter` that is mapped to the `iConverterFactory` factory class in the library:

```
=> CREATE LIBRARY iConverterLib as '/opt/vertica/sdk/examples/build/IconverterLib.so';
CREATE LIBRARY
=> CREATE FILTER Iconverter AS LANGUAGE 'C++' NAME 'IconverterFactory' LIBRARY IconverterLib;
CREATE FILTER FUNCTION
=> \x
Expanded display is on.
=> SELECT * FROM user_functions;
-[ RECORD 1 ]-----+-----
schema_name         | public
function_name       | Iconverter
procedure_type      | User Defined Filter
function_return_type |
function_argument_type |
function_definition |
volatility           |
is_strict            | f
```

is_fenced		f
comment		

See Also

- [CREATE LIBRARY](#)
- [DROP FILTER](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Load \(UDLs\)](#)

CREATE FUNCTION (SQL Functions)

Lets you store SQL expressions as functions in Vertica for use in queries. These functions are useful for executing complex queries or combining Vertica built-in functions. You simply call the function name you assigned.

Note: This topic describes how to use CREATE FUNCTION to create a SQL function. If you want to create a user-defined scalar function (UDSF), see [CREATE FUNCTION \(UDF\)](#).

In addition, if you want to see how to create a user-defined transform function (UDTF), see [CREATE TRANSFORM FUNCTION](#).

Syntax

```
CREATE [ OR REPLACE ] FUNCTION
... [schema.]function-name ( [ argname argtype [, ...] ] )
... RETURN rettype
... AS
... BEGIN
..... RETURN expression;
... END;
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>function-name</i>	<p>Specifies a name for the SQL function to create, where <i>function-name</i> conforms to conventions described in Identifiers. When using more than one schema, specify the schema that contains the function, as noted above.</p>
<i>argname</i>	<p>Specifies the name of the argument.</p>
<i>argtype</i>	<p>Specifies the data type for argument that is passed to the function. Argument types must match Vertica type names. See SQL Data Types.</p>
<i>rettype</i>	<p>Specifies the data type to be returned by the function.</p>
RETURN <i>expression</i> ;	<p>Specifies the SQL function (function body), which must be in the form of 'RETURN expression.' expression can contain built-in functions, operators, and argument names specified in the CREATE FUNCTION statement.</p> <p>A semicolon at the end of the expression is required.</p> <p>Note: Only one RETURN expression is allowed in the CREATE FUNCTION definition. FROM, WHERE, GROUP BY, ORDER BY, LIMIT, aggregation, analytics, and meta function are not allowed.</p>

Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.
- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.
- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

See [GRANT \(User Defined Extension\)](#) and [REVOKE \(User Defined Extension\)](#).

Notes

- A SQL function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.
- SQL Macros are flattened in all cases, including DDL.
- You can [create views](#) on the queries that use SQL functions and then query the views. When you create a view, a SQL function replaces a call to the user-defined function with the function body in a view definition. Therefore, when the body of the user-defined function is replaced, the view should also be replaced.
- If you want to change the body of a SQL function, use the `CREATE OR REPLACE` syntax. The command replaces the function with the new definition. If you change only the argument name or argument type, the system maintains both versions under the same function name. See **Examples** section below.
- If multiple SQL functions with same name and argument type are in the search path, the first match is used when the function is called.
- The strictness and volatility (stable, immutable, or volatile) of a SQL Macro are automatically inferred from the function's definition. Vertica then determines the correctness of usage, such as where an immutable function is expected but a volatile function is provided.
- You can return a list of all SQL functions by querying the system table `V_CATALOG.USER_FUNCTIONS` and executing the `vsql` meta-command `\df`. Users see only the functions on which they have `EXECUTE` privileges.

Example

This following statement creates a SQL function called `myzeroifnull` that accepts an `INTEGER` argument and returns an `INTEGER` result.

```
=> CREATE FUNCTION myzeroifnull(x INT) RETURN INT
    AS BEGIN
        RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
    END;
```

You can use the new SQL function (`myzeroifnull`) anywhere you use an ordinary SQL expression. For example, create a simple table:

```
=> CREATE TABLE tabwnulls(col1 INT);
=> INSERT INTO tabwnulls VALUES(1);
=> INSERT INTO tabwnulls VALUES(NULL);
=> INSERT INTO tabwnulls VALUES(0);
=> SELECT * FROM tabwnulls;
 a
 ---
 1
 0
 0
(3 rows)
```

Use the `myzeroifnull` function in a `SELECT` statement, where the function calls `col1` from table `tabwnulls`:

```
=> SELECT myzeroifnull(col1) FROM tabwnulls;
 myzeroifnull
 -----
            1
            0
            0
(3 rows)
```

Use the `myzeroifnull` function in the `GROUP BY` clause:

```
=> SELECT COUNT(*) FROM tabwnulls GROUP BY myzeroifnull(col1);
 count
 -----
      2
      1
(2 rows)
```

If you want to change a SQL function's body, use the `CREATE OR REPLACE` syntax. The following command modifies the `CASE` expression:

```
=> CREATE OR REPLACE FUNCTION zerowhennull(x INT) RETURN INT AS BEGIN
      RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
      END;
```

To see how this information is stored in the Vertica catalog, see [Viewing Information About SQL Functions](#) in *Extending Vertica*.

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [DROP FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)

- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Using User-Defined SQL Functions](#)

CREATE FUNCTION (UDF)

Adds a user-defined function (UDF) to the catalog. You must have already loaded this library using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Note: This topic describes how to use CREATE FUNCTION to create a User Defined Function. If you want to create a SQL function, see [CREATE FUNCTION \(SQL Function\)](#).

In addition, if you want to create a user-defined transform function (UDTF), see [CREATE TRANSFORM FUNCTION](#).

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [schema.]function-name  
... AS [ LANGUAGE 'language' ] NAME 'factory' LIBRARY library-name  
[ FENCED | NOT FENCED ];
```

Parameters

<code>OR REPLACE</code>	If you do not supply this parameter, the CREATE FUNCTION statement fails if an existing function matches the name and parameters of the function you are trying to define. If you do supply this parameter, the new function definition overwrites the old.
<code><i>schema</i></code>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<code><i>function-name</i></code>	The name of the function to create, where <i>function-name</i> conforms to conventions described in Identifiers . If the

	function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.
LANGUAGE ' <i>Language</i> '	<p>\The language used to develop this function, one of the following:</p> <ul style="list-style-type: none"> • C++ • Python • Java • R
NAME ' <i>factory</i> '	The name of the factory class in the shared library that generates the object to handle the function's processing.
LIBRARY <i>Library-name</i>	The name of the file that contains the C++ library, Python file, Java Jar file, or R functions file to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.
FENCED NOT FENCED	Enables or disables Fenced Mode for this function. Fenced mode is enabled by default. Functions written in Java and R always run in fenced mode.

Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.
- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.
- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

Notes

- The parameters and return value for the function are automatically determined by the CREATE FUNCTION statement, based on data supplied by the factory class.
- Multiple functions can share the same name if they have different parameters. When you call a multiply-defined function, Vertica selects the UDF function whose input parameters match the parameters in the function call to perform the processing. This behavior is similar to having multiple signatures for a method or function in other programming languages.
- You can return a list of all SQL functions and UDFs by querying the system table V_CATALOG.USER_FUNCTIONS or executing the vsql meta-command \df. Users see only the functions on which they have EXECUTE privileges.

Examples

The following example demonstrates loading a library named scalarfunctions, then defining a function named Add2ints that is mapped to the Add2intsInfo factory class in the library:

```
=> CREATE LIBRARY ScalarFunctions AS '/opt/vertica/sdk/examples/build/ScalarFunctions.so';
CREATE LIBRARY

=> CREATE FUNCTION Add2Ints AS LANGUAGE 'C++' NAME 'Add2IntsFactory' LIBRARY ScalarFunctions;
CREATE FUNCTION

=> \x
Expanded display is on.
=> SELECT * FROM USER_FUNCTIONS;

-[ RECORD 1 ]-----+-----
schema_name      | public
function_name    | Add2Ints
procedure_type   | User Defined Function
function_return_type | Integer
function_argument_type | Integer, Integer
function_definition | Class 'Add2IntsFactory' in Library 'public.ScalarFunctions'
volatility       | volatile
is_strict        | f
is_fenced        | t
comment          |

=> \x
Expanded display is off.
=> -- Try a simple call to the function
=> SELECT Add2Ints(23,19);
Add2Ints
-----
```

42
(1 row)

See Also

- [CREATE LIBRARY](#)
- [DROP FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Developing User-Defined Extensions \(UDxs\)](#)

CREATE PARSER

Adds a user-defined load parser function. You must have already loaded this library using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] PARSER [schema.]function-name  
... AS [ LANGUAGE 'language' ] NAME 'factory' LIBRARY library-name  
... [ FENCED | NOT FENCED ];
```

Parameters

<code>OR REPLACE</code>	If you do not supply this parameter, the CREATE PARSER statement fails if an existing function matches the name and parameters of the parser function you are trying to define. If you do supply this parameter, the new parser function definition overwrites the old.
<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example:

	<code>myschema.thisDBObject</code>
<i>function-name</i>	The name of the parser function to create. If the parser function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.
LANGUAGE ' <i>Language</i> '	The language used to develop this function, one of the following: <ul style="list-style-type: none"> • C++ • Java Default value: C++
NAME ' <i>factory</i> '	The name of the factory class in the shared library that generates the object to handle the parser function's processing. This is the same name used by the RegisterFactory class.
LIBRARY <i>library-name</i>	The name of the shared library that contains the C++ object to perform the processing for this parser function. This library must have been previously loaded using the CREATE LIBRARY statement.
FENCED NOT FENCED	Enables or disables Fenced Mode for this function. Default Value: FENCED

Notes

- The parameters and return value for the parser function are automatically determined by the CREATE PARSER statement, based on data supplied by the factory class.
- You can return a list of all SQL functions and User Defined Functions by querying the system table `V_CATALOG.USER_FUNCTIONS` or executing the vsql meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

Privileges

- Only a superuser can create or drop a function that uses a UDX library.
- To use a User Defined Parser, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined parser function. See [GRANT \(User Defined Extension\)](#) and [REVOKE \(User Defined Extension\)](#).

Important: Installing an untrusted UDL function can compromise the security of the server. UDX's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to UDX's to untrusted users.

Example

The following example demonstrates loading a library named BasicIntegerParserLib, then defining a function named BasicIntegerParser that is mapped to the BasicIntegerParserFactory factory class in the library:

```
=> CREATE LIBRARY BasicIntegerParserLib as '/opt/vertica/sdk/examples/build/BasicIntegerParser.so';
CREATE LIBRARY
=> CREATE PARSER BasicIntegerParser AS LANGUAGE 'C++' NAME 'BasicIntegerParserFactory' LIBRARY
BasicIntegerParserLib;
CREATE PARSER FUNCTION
=> \x
Expanded display is on.
=> SELECT * FROM user_functions;
-[ RECORD 1 ]-----+-----
schema_name         | public
function_name       | BasicIntegerParser
procedure_type      | User Defined Parser
function_return_type |
function_argument_type |
function_definition |
volatility          |
is_strict           | f
is_fenced           | f
comment             |
```

See Also

- [CREATE LIBRARY](#)
- [DROP PARSER](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Load \(UDLs\)](#)

CREATE SOURCE

Adds a user-defined load source function. You must have already loaded this library using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] SOURCE [schema.]function-name  
... AS LANGUAGE 'language' NAME 'factory' LIBRARY library-name  
... [ FENCED | NOT FENCED ];
```

Parameters

<code>OR REPLACE</code>	If you do not supply this parameter, the CREATE SOURCE statement fails if an existing function matches the name and parameters of the source function you are trying to define. If you do supply this parameter, the new source function definition overwrites the old.
<code><i>schema</i></code>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<code><i>function-name</i></code>	The name of the source function to create. If the source function

	name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.
LANGUAGE ' <i>Language</i> '	The language used to develop this function, one of the following: <ul style="list-style-type: none"> • C++ • Java Default value: C++
NAME ' <i>factory</i> '	The name of the factory class in the shared library that generates the object to handle the source function's processing. This is the same name used by the RegisterFactory class.
LIBRARY <i>library-name</i>	The name of the shared library that contains the C++ object to perform the processing for this source function. This library must have been previously loaded using the CREATE LIBRARY statement.
FENCED NOT FENCED	Enables or disables Fenced Mode for this function. Default Value: FENCED

Notes

- The parameters and return value for the source function are automatically determined by the CREATE SOURCE statement, based on data supplied by the factory class.
- You can return a list of all SQL functions and User Defined Functions by querying the system table `V_CATALOG.USER_FUNCTIONS` or executing the vsql meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

Privileges

- Only a superuser can create or drop a function that uses a UDX library.
- To use a User Defined Source, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined source function. See [GRANT \(User Defined Extension\)](#) and [REVOKE \(User Defined Extension\)](#).

Important: Installing an untrusted UDL function can compromise the security of the server. UDX's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to UDX's to untrusted users.

Example

The following example demonstrates loading a library named `curllib`, then defining a function named `curl` that is mapped to the `CurlSourceFactory` factory class in the library:

```
=> CREATE LIBRARY curllib as '/opt/vertica/sdk/examples/build/cURLLib.so';
CREATE LIBRARY
=> CREATE SOURCE curl AS LANGUAGE 'C++' NAME 'CurlSourceFactory' LIBRARY curllib;
CREATE SOURCE
=> \x
Expanded display is on.
=> SELECT * FROM user_functions;
-[ RECORD 1 ]-----+-----
schema_name          | public
function_name        | curl
procedure_type       | User Defined Source
function_return_type |
function_argument_type |
function_definition  |
volatility            |
is_strict             | f
is_fenced             | f
comment              |
```

See Also

- [CREATE LIBRARY](#)
- [DROP SOURCE](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Load \(UDLs\)](#)

CREATE TRANSFORM FUNCTION

Adds a user-defined transform function (UDTF) stored in a shared Linux library to the catalog. You must have already loaded this library using the [CREATE LIBRARY](#) statement. When you call the SQL function, Vertica passes the input table to the transform function in the library to process.

Note: This topic describes how to create a UDTF. To create a user-defined function (UDF), see [CREATE FUNCTION \(UDF\)](#). To create a SQL function, see [CREATE FUNCTION \(SQL\)](#).

Syntax

```
CREATE [OR REPLACE] TRANSFORM FUNCTION function-name
  AS [LANGUAGE 'Language' ] NAME 'factory'
  LIBRARY Library-name
  [ FENCED | NOT FENCED ];
```

Parameters

OR REPLACE	If you do not supply this parameter, the CREATE TRANSFORM FUNCTION statement fails if an existing function matches the name and parameters of the function you are trying to define. If you do supply this parameter, the new function definition overwrites the old.
<i>function-name</i>	The name to assign to the UDTF. This is the name you use in your SQL statements to call the function.
LANGUAGE ' <i>Language</i> '	The language used to develop this function, one of the following: <ul style="list-style-type: none">• C++• Java• R• PYTHON Default value: C++

NAME <i>'factory'</i>	The name of the factory class or R factory function in the shared library that generates the object to handle the function's processing.
LIBRARY <i>library-name</i>	The name of the shared library that contains the object to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.
FENCED NOT FENCED	Enables or disables Fenced Mode for this function. Functions written in R always run in fenced mode. Default Value: FENCED

Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.
- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.
- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

UDTF Query Restrictions

A query that includes a UDTF cannot contain:

- Any statements other than the [SELECT](#) statement containing the call to the UDTF and a PARTITION BY expression
- Any other [analytic function](#)
- A call to another UDTF
- A [TIMESERIES](#) clause
- A [pattern matching](#) clause
- A [gap filling and interpolation](#) clause

Notes

- The parameters and return values for the function are automatically determined by the CREATE TRANSFORM FUNCTION statement, based on data supplied by the factory class.
- You can assign multiple functions the same name if they have different parameters. When you call a multiply-defined function, Vertica selects the UDTF function whose input parameters match the parameters in the function call to perform the processing. This behavior is similar to having multiple signatures for a method or function in other programming languages.
- You can return a list of all UDTFs by querying the system table [V_CATALOG.USER_FUNCTIONS](#). You can only see functions for which you have EXECUTE privileges.

Examples

This example shows how to add a UDTF to the catalog.

```
=> CREATE TRANSFORM FUNCTION transFunct AS LANGUAGE 'C++' NAME 'myFactory' LIBRARY myFunction;
```

See Also

- [DROP FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Developing User-Defined Extensions \(UDxs\)](#)

CREATE HCATALOG SCHEMA

Define a schema for data stored in a Hive data warehouse using the HCatalog Connector. For more information, see [Using the HCatalog Connector](#) in Integrating with Apache Hadoop.

Most of the optional parameters are read out of Hadoop configuration files if available. See the Notes section for further information.

Syntax

```
CREATE HCATALOG SCHEMA [IF NOT EXISTS] schemaName
  [AUTHORIZATION user-id]
  [WITH
    [HOSTNAME='metastore-host']
    [PORT='metastore-port']
    [HIVESERVER2_HOSTNAME='hive-server-2-host']
    [WEBSERVICE_HOSTNAME='webHCat-host']
    [WEBSERVICE_PORT='webHCat-port']
    [WEBHDFS_ADDRESS='webhdfs-address']
    [HCATALOG_SCHEMA='hive-schema-name']
    [HCATALOG_USER='hcat-username']
    [HCATALOG_CONNECTION_TIMEOUT='timeout']
    [HCATALOG_SLOW_TRANSFER_LIMIT='xfer-limit']
    [HCATALOG_SLOW_TRANSFER_TIME='xfer-time'] ]
```

Parameters

Parameter	Description	Default Value
[IF NOT EXISTS]	If given, the statement exits without an error when the schema named in <i>schemaName</i> already exists.	N/A
<i>schemaName</i>	The name of the schema to create in the Vertica catalog. The tables in the Hive database will be available through this schema.	none
AUTHORIZATION <i>'user-id'</i>	The name of a Vertica account to own the schema being created. This parameter is ignored if Kerberos authentication is being used; in that case the current vsql user is used.	current username
HOSTNAME= <i>'metastore-host'</i>	The hostname or IP address of the database server that stores the Hive data warehouse's metastore information. If this value is not specified, hive-site.xml must be available.	none
PORT= <i>'metastore-port'</i>	The port number on which the metastore database is running.	See Notes

<p>HIVESERVER2_ HOSTNAME= <i>'hiveserver2- host'</i></p>	<p>The hostname or IP address of the HiveServer2 service. This parameter is optional if in hive-site.xml you set one of the following properties:</p> <ul style="list-style-type: none"> • hive.server2.thrift.bind.host to a valid host • hive.server2.support.dynamic.service.discovery to true <p>This parameter is ignored if you are using WebHCat.</p>	<p>none</p>
<p>WEBSERVICE_ HOSTNAME= <i>'webHCat-host'</i></p>	<p>The hostname or IP address of the WebHCat service, if using WebHCat instead of HiveServer2. If this value is not specified, webhcat-site.xml must be available.</p>	<p>none</p>
<p>WEBSERVICE_ PORT= <i>'webHCat-port'</i></p>	<p>The port number on which the WebHCat service is running, if using WebHCat instead of HiveServer2. If this value is not specified, webhcat-site.xml must be available.</p>	<p>See Notes</p>
<p>WEBHDFS_ ADDRESS= <i>'webhdfs- address'</i></p>	<p>The host and port ("host:port") for the WebHDFS service. This parameter is used only for reading ORC and Parquet files. If this value is not set, hdfs-site.xml must be available to read these file types through the HCatalog Connector.</p>	<p>none</p>
<p>HCATALOG_ SCHEMA= <i>'hive-schema- name'</i></p>	<p>The name of the Hive schema or database that the Vertica schema is being mapped to.</p>	<p><i>schemaName</i></p>
<p>HCATALOG_USER= <i>'hcat-username'</i></p>	<p>The username of the HCatalog user to use when making calls to the HiveServer2 or WebHCat server.</p>	<p>current username</p>
<p>HCATALOG_ CONNECTION_ TIMEOUT= <i>'timeout'</i></p>	<p>The number of seconds the HCatalog Connector waits for a successful connection to the HiveServer or WebHCat server. A value of 0 means wait indefinitely.</p>	<p>See Notes</p>
<p>HCATALOG_SLOW_ TRANSFER_ LIMIT=</p>	<p>The lowest data transfer rate (in bytes per second) from the HiveServer2 or WebHCat server that the HCatalog Connector accepts. See <i>xfer-time</i> for</p>	<p>See Notes</p>

' <i>xfer-limit</i> '	details.	
HCATALOG_SLOW_TRANSFER_TIME= ' <i>xfer-time</i> '	The number of seconds the HCatalog Connector waits before enforcing the data transfer rate lower limit. After this time has passed, the HCatalog Connector tests whether the data transfer rate is at least as fast as the value set in <i>xfer-limit</i> . If it is not, then the HCatalog Connector breaks the connection and terminates the query.	See Notes

Notes

The default values for *timeout*, *xfer-limit*, and *xfer-time* are set by the configuration parameters HCatConnectionTimeout, HCatSlowTransferLimit, and HCatSlowTransferTime. See [Hadoop Parameters](#) in the Administrator's Guide for more information.

If you are using Kerberos authentication, the current user name is always used and the AUTHORIZATION parameter is ignored.

When using HiveServer2 (the default), use HIVESERVER2_HOSTNAME to specify the server host. When using WebHCat, use WEBSERVICE_HOSTNAME to specify the server host.

If you copied the Hadoop configuration files as described in [Configuring Vertica for HCatalog](#), you can omit most parameters. By default this statement uses the values specified in those configuration files, though you can still override them with this statement. If the configuration files are complete, the following is a valid statement:

```
=> CREATE HCATALOG SCHEMA hcat;
```

If a value is not specified in the configuration files and a default is shown in the parameter list, then that default value is used.

Privileges

The user must be a superuser or be granted all permissions on the database to use CREATE HCATALOG SCHEMA.

Example

The following example shows how to use CREATE HCATALOG SCHEMA to define a new schema for tables stored in a Hive database and then query the system tables that contain information about those tables:

```
=> CREATE HCATALOG SCHEMA hcat WITH HOSTNAME='hcat'
      HCATALOG_SCHEMA='default' HIVESERVER2_HOSTNAME='hs.example.com'
      HCATALOG_USER='admin';
WARNING 0: HOSTNAME will be ignored. hive-site.xml must be contain property [hive.metastore.uris],
or both HOSTNAME and PORT must be specified
CREATE SCHEMA
=> \x
Expanded display is on.

=> SELECT * FROM v_catalog.hcatalog_schemata;
-[ RECORD 1 ]-----+-----
schema_id          | 45035996273748224
schema_name        | hcat
schema_owner_id    | 45035996273704962
schema_owner       | admin
create_time        | 2017-05-22 12:04:59.692155-04
hostname           | hcat
port               | -1
hiveserver2_hostname | hs.example.com
webservice_hostname |
webservice_port    | 50111
webhdfs_address    | hs.example.com:50070
hcatalog_schema_name | default
hcatalog_user_name  | admin
hcatalog_connection_timeout | -1
hcatalog_slow_transfer_limit | -1
hcatalog_slow_transfer_time | -1

=> SELECT * FROM v_catalog.hcatalog_table_list;
-[ RECORD 1 ]-----+-----
table_schema_id    | 45035996273748224
table_schema       | hcat
hcatalog_schema    | default
table_name         | nation
hcatalog_user_name | admin
-[ RECORD 2 ]-----+-----
table_schema_id    | 45035996273748224
table_schema       | hcat
hcatalog_schema    | default
table_name         | raw
hcatalog_user_name | admin
-[ RECORD 3 ]-----+-----
table_schema_id    | 45035996273748224
table_schema       | hcat
hcatalog_schema    | default
table_name         | raw_rcfile
hcatalog_user_name | admin
-[ RECORD 4 ]-----+-----
table_schema_id    | 45035996273748224
```

```
table_schema      | hcat
hcatalog_schema   | default
table_name        | raw_sequence
hcatalog_user_name | admin
```

CREATE LIBRARY

Loads a library containing User Defined Extensions (UDxs) into the Vertica catalog. After loading a library in the catalog, you can use statements such as CREATE FUNCTION to define the extensions contained in the library. See [Developing User-Defined Extensions \(UDxs\)](#) in Extending Vertica for details.

Warning: You can choose to run UDxs developed in C++ in unfenced mode. Unfenced UDxs run directly in the Vertica process. If the UDx you run in unfenced mode has bugs, it can negatively impact the database, causing instability or even crashes. To avoid these issues, run your UDx only in fenced mode. A few UDx types can only be run in unfenced mode.

Syntax

```
CREATE [OR REPLACE] LIBRARY
  [schema.]Library-name
  AS 'Library-path'
  [ DEPENDS 'support-path' ]
  [ LANGUAGE 'Language' ]
```

Parameters

OR REPLACE	Replaces the old library with the new one. If you do not supply this parameter, the CREATE LIBRARY statement fails when an existing library matches the name the library you are trying to define.
<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>Library-name</i>	A name to assign to this library, where <i>library-name</i> conforms to conventions described in Identifiers . Use this name in a CREATE FUNCTION statement to enable user defined functions stored in the

	<p>library.</p> <p>Best practice:</p> <p>This name does not have to match the library file name. However, consider using the same name to avoid confusion.</p>
<i>Library-path</i>	The absolute path and file name of the library to load. This file must be located in the initiator node's filesystem.
DEPENDS ' <i>support-path</i> '	<p>Indicates that the UDx library depends on one or more support libraries.</p> <p>Valid values: One or more absolute paths to the support libraries files, located in the initiator node's filesystem. Separate multiple paths with colons (:). Specify a directory containing multiple libraries using an asterisk wildcard (*). For example: '/home/mydir/mylibs/'.</p>
LANGUAGE ' <i>Language</i> '	<p>The programming language used to develop the function, where <i>language</i> is one of the following:</p> <ul style="list-style-type: none"> • C++ • Python • Java • R

Privileges

superuser

Usage Considerations

- As part of the loading process, Vertica distributes the library file and any supporting libraries to all nodes in the database. cluster. Any nodes that are down or that are added to the cluster later automatically receive a copy of the files when they join the cluster.
- Vertica makes its own copies of the library files. Later modification or deletion of the original files specified in the statement does not effect the library defined in the catalog. To update the library, use the ALTER LIBRARY statement.

- Simply loading the library is no guarantee that it functions correctly. CREATE LIBRARY performs some basic checks on the library file to verify it is compatible with Vertica. The statement fails if it detects that the library was not correctly compiled or it finds other basic incompatibilities. However, there are many issues in shared libraries that CREATE LIBRARY cannot detect.
- Libraries are added to the database catalog, and therefore persist across database restarts.
- If your Java library depends on native libraries (SO files), use DEPENDS to specify the path and call System.loadLibrary() in your UDX to load the native libraries from that path.

Examples

To load a library in the home directory of the dbadmin account with the name MyFunctions:

```
=> CREATE LIBRARY MyFunctions AS 'home/dbadmin/my_functions.so';
```

To load a library located in the directory where you started vsql:

```
=> \set libfile '`pwd`'/MyOtherFunctions.so\';  
=> CREATE LIBRARY MyOtherFunctions AS :libfile;
```

To load a Java library named JavaLib.jar that depends on multiple support JAR files in the /home/dbadmin/mylibs subdirectory:

```
=> CREATE LIBRARY DeleteVowelsLib AS '/home/dbadmin/JavaLib.jar'  
DEPENDS '/home/dbadmin/mylibs/*' LANGUAGE 'JAVA';
```

See Also

- [DROP LIBRARY](#)
- [ALTER LIBRARY](#)
- [CREATE FUNCTION \(UDF\)](#)

CREATE LOCAL TEMPORARY VIEW

Creates or replaces a local temporary view. Views are read only, so they do not support insert, update, delete, or copy operations. Local temporary views are session-scoped, so they are

visible only to their creator in the current session. Vertica drops the view when the session ends.

Note: Vertica does not support global temporary views.

Syntax

```
CREATE [OR REPLACE] LOCAL TEMP[ORARY] VIEW view-name [ (column-name[, ...] ) ]  
AS query ]
```

Parameters

<code>OR REPLACE</code>	Specifies to overwrite the existing view <i>view-name</i> . If you omit this option and <i>view-name</i> already exists, <code>CREATE VIEW</code> returns an error.
<i>view-name</i>	Specifies the name of the view to create, where <i>view-name</i> conforms to conventions described in Identifiers . <i>view-name</i> must not be the name of an existing table, view, or projection.
<i>column-name</i> [, ...]	A list of names to use as view column names. Vertica maps view column names to query columns according to the order of their respective lists. By default, the view uses column names as they are specified in the query. Each view can contain up to 1600 columns.
<i>query</i>	A SELECT statement that the temporary view executes. The <code>SELECT</code> statement can reference tables, temporary tables, and other views.

Privileges

See [Creating Views](#)

Example

The following `CREATE LOCAL TEMPORARY VIEW` statement creates the temporary view `myview`. This view sums all individual incomes of customers listed in the `store.store_sales_fact` table, and groups results by state:

```
=> CREATE LOCAL TEMP VIEW myview AS
  SELECT SUM(annual_income), customer_state FROM public.customer_dimension
  WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact)
  GROUP BY customer_state
  ORDER BY customer_state ASC;
```

The following example uses the temporary view `myview` with a `WHERE` clause that limits the results to combined salaries greater than \$2 billion:

```
=> SELECT * FROM myview WHERE SUM > 2000000000;
```

SUM	customer_state
2723441590	AZ
29253817091	CA
4907216137	CO
3769455689	CT
3330524215	FL
4581840709	IL
3310667307	IN
2793284639	MA
5225333668	MI
2128169759	NV
2806150503	PA
2832710696	TN
14215397659	TX
2642551509	UT

(14 rows)

See Also

- [ALTER VIEW](#)
- [CREATE VIEW](#)
- [Creating Views](#)

CREATE LOCATION

Creates a new storage location where Vertica can store data. After you create the location, you create storage policies that assign the storage location to the database objects that will store data in the location.

Cautions

While no technical issue prevents you from using `CREATE LOCATION` to add one or more Network File System (NFS) storage locations, Vertica does not support NFS data or catalog storage except for MapR mount points. You will be unable to run queries against any other NFS data. When creating locations on MapR file systems, you must specify `ALL NODES SHARED`.

If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

Syntax

```
CREATE LOCATION 'path'  
  [NODE 'nodename' | ALL NODES]  
  [SHARED]  
  [USAGE 'usetype']  
  [LABEL 'LabelName']
```

Arguments

<i>path</i>	<p>Where Vertica will store this location's data. The type of filesystem on which the location is based determines the format of this argument:</p> <ul style="list-style-type: none">• For storage locations on the Linux filesystem, <i>path</i> must be an absolute path to the directory where Vertica can write the storage location's data.• For storage locations on HDFS, <i>path</i> must be a WebHDFS URI where Vertica can write the storage location's data. See more information below.
[NODE <i>nodename</i> ALL NODES]	<p>The node or nodes on which the storage location is defined.</p> <ul style="list-style-type: none">• NODE — Use this keyword to create the storage location on a single node. Specify the node using its name as it appears in the NODES system table.• ALL NODES—Use this keyword to create the storage location on all

	<p>nodes</p> <p>Default Value: ALL NODES</p>
[SHARED]	<p>Indicates the location set by the <i>path</i> is shared (used by all of the nodes) rather than local to each node. See below for details.</p>
[USAGE 'usetype']	<p>The type of data the storage location can hold.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • 'TEMP'—Vertica uses the location to store temporary files it creates while processing queries. • 'DATA'—The storage location can only store data. • 'TEMP,DATA'—The storage location can store both temporary files and data. • 'USER'—Users who have been granted access to the storage location can read and store data there (see GRANT (Storage Location)). This usage type can be used with external tables (see CREATE EXTERNAL TABLE AS COPY), which by default are readable only by administrators. <p>Default Value: 'TEMP,DATA'</p>
[LABEL ' <i>LabelName</i> ']	<p>A label for the storage location. You use this name later when assigning the storage location to data objects.</p>

Shared vs. Local Storage

The SHARED keyword indicates that the location set by the *path* argument is shared by all nodes. Most remote filesystems (such as HDFS) are shared. For these filesystems, the *path* argument represents a single location where all of the nodes store data. Each node creates its own subdirectory to hold its own files in a shared storage location. These subdirectories prevent the nodes from overwriting each other's files. Even if your cluster has only one node, you must include the SHARED keyword if you are using a remote filesystem. If the location is declared as USER Vertica does not create sub directories for each node. The setting of USER takes precedence over SHARED.

If you do not supply this keyword, the new storage location is local. The *path* argument specifies a location that is unique for each node in the cluster. This location is usually a path in the node's own filesystem. Storage locations contained in filesystems that are local to each node (such as the Linux filesystem) are always local.

WebHDFS URIs

The URIs you supply in the *path* argument are similar to the URLs for accessing files through WebHDFS, with a few differences:

- The protocol for the URI is `webhdfs://` rather than `http://`.
- The path portion of the URI does not include the `/webhdfs/v1/` portion of the path. Instead, after the hostname and port, specify the HDFS path from the root directory.

For example, suppose you can access the HDFS directory you want to use as a storage location using the URL `http://hadoop:50070/webhdfs/v1/user/dbadmin`. Then the URI you supply to the CREATE LOCATION statement is `webhdfs://hadoop:50070/user/dbadmin`.

For more information about WebHDFS URIs, see the [WebHDFS REST API](#) page.

Privileges

The user must be a superuser to use CREATE LOCATION.

In addition, the Vertica process must have read and write permissions to the location where data will be stored. Each type of filesystem has its own requirements:

- Linux—The database administrator account (usually named `dbadmin`) must have full read and write access to the directory in the *path* argument.
- HDFS without Kerberos—Requires a Hadoop user whose username matches the Vertica database administrator username (usually `dbadmin`). This Hadoop user must have read and write access to the HDFS directory specified in the *path* argument.
- HDFS with Kerberos—Requires a Hadoop user whose username matches the principal in the keytab file on each Vertica node. This is *not* the same as the database administrator username. This Hadoop user must have read and write access to the HDFS directory stored in the *path* argument.

Examples

The following example shows how to create a storage location in the local Linux filesystem for temporary data storage.

```
=> CREATE LOCATION '/home/dbadmin/testloc' USAGE 'TEMP' LABEL 'tempfiles';
```

The following example shows how to create a storage location on the HDFS cluster available from the Hadoop name node `hadoop.example.com` in the `/user/dbadmin` directory. The HDFS cluster does not use Kerberos.

```
=> CREATE LOCATION 'webhdfs://hadoop.example.com:50070/user/dbadmin' ALL NODES SHARED  
USAGE 'data' LABEL 'coldstorage';
```

The following example shows how to create the same storage location, but on a Hadoop cluster that uses Kerberos. Note the output that reports the principal being used.

```
=> CREATE LOCATION 'webhdfs://hadoop.example.com:50070/user/dbadmin' ALL NODES SHARED  
USAGE 'data' LABEL 'coldstorage';  
NOTICE 0: Performing HDFS operations using kerberos principal [vertica/hadoop.example.com]  
CREATE LOCATION
```

The following example shows how to create a location for user data, grant access to it, and use it to create an external table.

```
=> CREATE LOCATION '/tmp' ALL NODES USAGE 'user';  
CREATE LOCATION  
=> GRANT ALL ON LOCATION '/tmp' to Bob;  
GRANT PRIVILEGE  
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/data/ext1.dat' DELIMITER ',';  
CREATE TABLE
```

See Also

- [Managing Storage Locations](#) in the Administrator's Guide
- [Vertica Storage Location for HDFS](#) in Integrating with Apache Hadoop.
- [ALTER_LOCATION_LABEL](#)
- [ALTER_LOCATION_USE](#)

- [DROP_LOCATION](#)
- [SET_OBJECT_STORAGE_POLICY](#)

CREATE NETWORK INTERFACE

Identifies a network interface to which a node belongs. Use this statement when you want to configure [import/export](#) operations from individual nodes to other Vertica clusters.

Syntax

```
CREATE NETWORK INTERFACE network-interface-name ON node-name WITH 'node-IP-address'
```

<i>network-interface-name</i>	The name you assign to the network interface, where <i>network-interface-name</i> conforms to conventions described in Identifiers .
<i>node-name</i>	The name of the node.
<i>node-IP-address</i>	The node's IP address, either a public or private IP address. For more information, see Using Public and Private IP Networks .

Privileges

Superuser

Examples

Create a network interface:

```
=> CREATE NETWORK INTERFACE mynetwork ON v_vmart_node0001 WITH '123.4.5.6';
```

CREATE NOTIFIER

Creates a push-based notifier to send event notifications and messages out of Vertica.

Syntax

```
CREATE NOTIFIER notifier-name ACTION action-url MAXMEMORYSIZE max-memory-size
... [ [NO] CHECK COMMITTED ]
... [ ENABLE | DISABLE ]
... [ IDENTIFIED BY uuid ]
... [ MAXPAYLOAD max-payload-size ]
... [ PARAMETERS 'adapter-params' ]
```

Parameters

<i>notifier-name</i>	This notifier's unique identifier.
ACTION <i>action-url</i>	Identifies the target Kafka server, where <i>action-url</i> has the following format: <i>kafka://kafka-server-ip-address:port-number</i> For example: <i>kafka://127.0.0.1:9092</i>
MAXMEMORYSIZE	The maximum size of the internal notifier, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows: <i>MAXMEMORYSIZE integer{K M G T}</i> If the queue exceeds this size, the notifier drops excess messages.
[NO] CHECK COMMITTED	Specifies to wait for delivery confirmation before sending the next message in the queue. Not all messaging systems support delivery confirmation.
ENABLE DISABLE	Specifies whether to enable or disable the notifier. If you omit this parameter, Vertica sets this notifier to ENABLE.
IDENTIFIED BY <i>uuid</i>	Specifies the notifier's unique identifier. If set, all the

	messages published by this notifier have this attribute.
MAXPAYLOAD	The maximum size of the message, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows: <code>MAXPAYLOAD integer{K M G T}</code> The default setting is adapter-specific—for example, 1 M for Kafka.
PARAMETERS ' <i>adapter-params</i> '	Specifies one or more optional adapter parameters that are passed as a string to the adapter. Adapter parameters apply only to the adapter associated with the notifier. For Kafka notifiers, refer to Configuring Kafka for Vertica .

Privileges

Database Administrator

Examples

Create a Kafka notifier:

```
=> CREATE NOTIFIER my_dc_notifier
      ACTION 'kafka://172.16.20.10:9092'
      MAXMEMORYSIZE '1G'
      IDENTIFIED BY 'f8b0278a-3282-4e1a-9c86-e0f3f042a971'
      NO CHECK COMMITTED;
```

Create a notifier with an adapter-specific parameter:

```
=> CREATE NOTIFIER my_notifier
      ACTION 'kafka://127.0.0.1:9092'
      MAXMEMORYSIZE '10M'
      PARAMETERS 'queue.buffering.max.ms=1000';
```

See Also

- [ALTER NOTIFIER](#)
- [DROP NOTIFIER](#)

- [Monitoring Vertica Using Notifiers](#) in the Administrator's Guide.

CREATE PROCEDURE

Adds an external procedure to Vertica. See [Implementing External Procedures](#) in Extending Vertica for more information about external procedures.

Syntax

```
CREATE PROCEDURE [schema.]procedure-name (  
... [ argname ] [ argtype [,...] ] )  
... AS 'exec-name'  
... LANGUAGE 'Language-name'  
... USER 'OS-user'
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>procedure-name</i>	Specifies a name for the external procedure, where <i>procedure-name</i> conforms to conventions described in Identifiers .
<i>argname</i>	[Optional] Presents a descriptive argument name to provide a cue to procedure callers.
<i>argtype</i>	[Optional] Specifies the data type for argument(s) that will be passed to the procedure. Argument types must be one of the following Vertica type names: BIGINT, BOOLEAN, DECIMAL, DOUBLE PRECISION, FLOAT, FLOAT8, INT, INT8, INTEGER, MONEY, NUMBER, NUMERIC, REAL, SMALLINT, TINYINT, VARCHAR.
AS	Specifies the executable program in the procedures directory.
LANGUAGE	Specifies the procedure language. This parameter must be set to EXTERNAL.
USER	Specifies the user executed as. The user is the owner of the file. The

external program must allow execute privileges for this user. The user cannot be root.

Privileges

To create a procedure a superuser must have CREATE privilege on schema to contain procedure.

Notes

- A procedure file must be owned by the database administrator (OS account) or by a user in the same group as the administrator. (The procedure file owner cannot be root.) The procedure file must also have the set UID attribute enabled, and allow read and execute permission for the group.
- By default, only a database superuser can execute procedures. However, a superuser can grant the right to execute procedures to other users. See [GRANT \(Procedure\)](#).

Important: External procedures that you create with [CREATE PROCEDURE](#) are always run with Linux dbadmin privileges. If a dbadmin or pseudosuperuser grants a non-dbadmin permission to run a procedure using [GRANT \(Procedure\)](#), be aware that the non-dbadmin user runs the procedure with full Linux dbadmin privileges.

Example

This example illustrates how to create a procedure named *helloplanet* for the *helloplanet.sh* external procedure file. This file accepts one varchar argument.

Sample file:

```
#!/bin/bash
echo "hello planet argument: $1" >> /tmp/myprocedure.log
```

Issue the following SQL to create the procedure:

```
=> CREATE PROCEDURE helloplanet(arg1 varchar) AS 'helloplanet.sh' LANGUAGE 'external' USER 'dbadmin';
```

See Also

- [DROP PROCEDURE](#)
- [Installing External Procedure Executable Files](#)

CREATE PROFILE

Creates a [profile](#) that controls password requirements for users.

Syntax

```
CREATE PROFILE name LIMIT
... [PASSWORD_LIFE_TIME {life-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_GRACE_TIME {grace-period | DEFAULT | UNLIMITED}]
... [FAILED_LOGIN_ATTEMPTS {login-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_LOCK_TIME {lock-period | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_MAX {reuse-limit | DEFAULT | UNLIMITED}]
... [PASSWORD_REUSE_TIME {reuse-period | DEFAULT | UNLIMITED}]
... [PASSWORD_MAX_LENGTH {max-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LENGTH {min-length | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LETTERS {min-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_UPPERCASE_LETTERS {min-cap-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_LOWERCASE_LETTERS {min-lower-letters | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_DIGITS {min-digits | DEFAULT | UNLIMITED}]
... [PASSWORD_MIN_SYMBOLS {min-symbols | DEFAULT | UNLIMITED}]
```

Note: For all parameters, the special DEFAULT value means that the parameter's value is inherited from the DEFAULT profile. Any changes to the parameter in the DEFAULT profile is reflected by all of the profiles that inherit that parameter. Any parameter not specified in the CREATE PROFILE command is set to DEFAULT.

Parameters

Name	Description	Meaning of UNLIMITED value
<i>name</i>	The name of the profile to create, where <i>name</i> conforms to	N/A

Name	Description	Meaning of UNLIMITED value
	conventions described in Identifiers .	
PASSWORD_LIFE_TIME <i>Life-Limit</i>	Integer number of days a password remains valid. After the time elapses, the user must change the password (or will be warned that their password has expired if PASSWORD_GRACE_TIME is set to a value other than zero or UNLIMITED).	Passwords never expire.
PASSWORD_GRACE_TIME <i>grace-period</i>	Integer number of days the users are allowed to login (while being issued a warning message) after their passwords are older than the PASSWORD_LIFE_TIME. After this period expires, users are forced to change their passwords on login if they have not done so after their password expired.	No grace period (the same as zero)
FAILED_LOGIN_ATTEMPTS <i>Login-Limit</i>	Number of consecutive failed login attempts permitted before locking a user's account.	Accounts are never locked, no matter how many failed login attempts are made.
PASSWORD_LOCK_TIME <i>Lock-period</i>	Integer value setting the number of days an account is locked after a user's account is locked after too many failed login attempts. After the PASSWORD_LOCK_TIME has expired, the account is automatically unlocked.	Accounts locked because of too many failed login attempts are never automatically unlocked. They must be manually unlocked by the database

Name	Description	Meaning of UNLIMITED value
		superuser.
PASSWORD_REUSE_MAX <i>reuse-limit</i>	The number of password changes that need to occur before the current password can be reused.	Users are not required to change passwords a certain number of times before reusing an old password.
PASSWORD_REUSE_TIME <i>reuse-period</i>	The integer number of days that must pass after a password has been set before it can be reused.	Password reuse is not limited by time.
PASSWORD_MAX_LENGTH <i>max-length</i>	The maximum number of characters allowed in a password. Value must be in the range of 8 to 100.	Passwords are limited to 100 characters.
PASSWORD_MIN_LENGTH <i>min-length</i>	The minimum number of characters required in a password. Valid range is 0 to <i>max-length</i> .	Equal to <i>max-length</i> .
PASSWORD_MIN_LETTERS <i>min-of-letters</i>	Minimum number of letters (a-z and A-Z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_UPPERCASE_LETTERS <i>min-cap-letters</i>	Minimum number of capital letters (A-Z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_LOWERCASE_LETTERS <i>min-lower-letters</i>	Minimum number of lowercase letters (a-z) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).
PASSWORD_MIN_DIGITS <i>min-digits</i>	Minimum number of digits (0-9) that must be in a password. Valid range is 0 to <i>max-length</i> .	0 (no minimum).

Name	Description	Meaning of UNLIMITED value
PASSWORD_MIN_SYMBOLS <i>min-symbols</i>	Minimum number of symbols (any printable non-letter and non-digit character, such as \$, #, @, and so on) that must be in a password. Valid range is 0 to <i>max-Length</i> .	0 (no minimum).

Privileges

Must be a superuser to create a profile.

Note: Only the profile settings for how many failed login attempts trigger [Account Locking](#) and how long accounts are locked have an effect on password authentication methods such as LDAP or GSS. All password [complexity](#), reuse, and [lifetime settings](#) affect only passwords that Vertica manages.

Example

```
=> CREATE PROFILE sample_profile LIMIT PASSWORD_MAX_LENGTH 20;
```

See Also

- [ALTER PROFILE](#)
- [DROP PROFILE](#)
- [Creating a Database Name and Password](#)

CREATE PROJECTION

Creates metadata for a projection in the Vertica catalog.

Note: For detailed information about using CREATE PROJECTION to create live aggregate projections and Top-K projections, see CREATE PROJECTION ([Live Aggregate Projections](#)). To create live aggregate projections that support user-defined transform functions, see CREATE PROJECTION ([UDTFs](#)).

Syntax

```
CREATE PROJECTION [ IF NOT EXISTS ]
...projection-name
...[ (
.....{ projection-col | grouped-clause
..... [ ENCODING encoding-type ]
..... [ ACCESSRANK integer ]
.....}[,...]
...) ]
AS SELECT
...select-list from-clause
...[ ORDER BY column-expr[,...] ]
...[ segmentation-spec ]
...[ KSAFE [ k-num ] ]
```

Parameters

<p>IF NOT EXISTS</p>	<p>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.</p> <p>The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.</p> <p>For related information, see ON_ERROR_STOP.</p>
<p><i>projection-name</i></p>	<p>The name of the projection to create. The projection is created in the same schema as the anchor table.</p> <p>If the projection is segmented, Vertica uses this string as the</p>

	projection base name when it creates unique identifiers for buddy projections. For more information, see Projection Naming in the Administrator's Guide.
<i>projection-col</i>	The name of a projection column. The list of projection columns must match the <i>select-list</i> columns and expressions in number, type, and sequence. If projection column names are omitted, Vertica uses the anchor table column names specified in <i>select-list</i> .
<i>grouped-clause</i>	See GROUPED Clause .
ENCODING <i>encoding-type</i>	Specifies the column encoding type , by default set to AUTO.
ACCESSRANK <i>integer</i>	Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking .
<i>select-list</i>	Specifies the columns or column expressions to select from one or more tables, one of the following: <ul style="list-style-type: none"> • * (asterisk) Lists all columns in the queried tables. • <i>expression</i> [[AS] <i>output-name</i>] [, ...] A table column or column expression to select from the queried tables. You can optionally qualify <i>expression</i> with an output name, which can be used in two ways: <ul style="list-style-type: none"> ■ Label the column for display. ■ Refer to the column in the projection's ORDER BY clause.
<i>from-clause</i>	A comma-separated list of data sources to query.
ORDER BY	Specifies columns from the SELECT list on which to sort the projection. The ORDER BY clause cannot include qualifiers ASC or DESC. Vertica always stores projection data in ascending sort order. If you omit the ORDER BY clause, Vertica uses <i>select-list</i> to sort the projection.

<p><i>segmentation-spec</i></p>	<p>Specifies how to distribute projection data with one of the following clauses:</p> <ul style="list-style-type: none"> • <i>hash-segmentation-clause</i>: Specifies to segment projection data evenly and distribute across cluster nodes. Vertica recommends segmenting large tables. • <i>unsegmented-clause</i>: Specifies to create an unsegmented projection. <p>If the anchor table and projection both omit specifying segmentation, the projection is defined with a hash segmentation clause that includes all columns in the SELECT list , as follows:</p> <pre>SEGMENTED BY HASH(<i>column-expr</i>[, ...]) ALL NODES OFFSET 0;</pre>
<p>KSAFE [<i>k-num</i>]</p>	<p>Specifies K-safety for the projection, where <i>k-num</i> must be equal to or greater than system K-safety. Vertica ignores this parameter if set for unsegmented projections. If you omit <i>k-num</i>, Vertica uses system K-safety.</p> <p>Vertica sets projection K-safety as follows:</p> <ul style="list-style-type: none"> • KSAFE and OFFSET clause omitted: Uses system K-safety. • OFFSET clause is omitted and KSAFE is specified: Uses KSAFE setting. • KSAFE is omitted and the OFFSET clause is specified: Uses OFFSET setting. <p>If the CREATE PROJECTION statement specifies KSAFE and the OFFSET clause, Vertica returns an error.</p> <p>For general information, see K-Safety in Vertica Concepts.</p>

Privileges

See [Projection Privileges](#).

Requirements

- To prevent data loss and inconsistencies, tables must have at least one superprojection. You cannot drop a projection if that projection is the table's only superprojection.
- You cannot drop a buddy projection if dropping that projection violates system K-safety.

Creating Projections with Expressions

The following example shows a projection that calculates the product of two numbers. The anchor table is defined as follows:

```
=> CREATE TABLE values (a INT, b INT);
```

To create a projection that calculates the product of a and b, use a statement like the following:

```
=> CREATE PROJECTION values_product (a, b, product_value)  
AS SELECT a, b, a*b FROM values  
SEGMENTED BY HASH(a) ALL NODES KSAFE;
```

To query that projection, you must use the name that Vertica assigned to it:

```
=> SELECT * FROM values_product_b0;
```

or

```
=> SELECT * FROM values_product_b1;
```

Grouping Correlated Columns

The following example shows how to group highly correlated columns `bid` and `ask`. The `stock` column is stored separately.

```
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);  
=> CREATE PROJECTION tradeproj (stock ENCODING RLE,  
GROUPED(bid ENCODING DELTAVAL, ask))  
AS (SELECT * FROM trades) KSAFE 1;
```

The following example show how to create a projection that uses expressions in the column definition. The projection contains two integer columns `a` and `b`, and a third column `product_value` that stores the product of `a` and `b`:

```
=> CREATE TABLE values (a INT, b INT  
=> CREATE PROJECTION product (a, b, product_value) AS  
SELECT a, b, a*b FROM values ORDER BY a KSAFE;
```

See Also

[Working with Projections](#)

Encoding Types

Vertica supports various encoding and compression types, specified by the following ENCODING parameter arguments:

- [AUTO \(default\)](#)
- [BLOCK_DICT](#)
- [BLOCKDICT_COMP](#)
- [BZIP_COMP](#)
- [COMMONDELTA_COMP](#)
- [DELTARANGE_COMP](#)
- [DELTAVAL](#)
- [GCDDELTA](#)
- [GZIP_COMP](#)
- [RLE](#)

AUTO (default)

AUTO encoding is ideal for sorted, many-valued columns such as primary keys. It is also suitable for general purpose applications for which no other encoding or compression scheme is applicable. Therefore, it serves as the default if no encoding/compression is specified.

Column data type	Default encoding type
------------------	-----------------------

BINARY/VARBINARY BOOLEAN CHAR/VARCHAR FLOAT	Lempel-Ziv-Oberhumer-based (LZO) compression
DATE/TIME/TIMESTAMP INTEGER INTERVAL	Compression scheme based on the delta between consecutive column values.

The CPU requirements for this type are relatively small. In the worst case, data might expand by eight percent (8%) for LZO and twenty percent (20%) for integer data.

BLOCK_DICT

For each block of storage, Vertica compiles distinct column values into a dictionary and then stores the dictionary and a list of indexes to represent the data block.

BLOCK_DICT is ideal for few-valued, unsorted columns where saving space is more important than encoding speed. Certain kinds of data, such as stock prices, are typically few-valued within a localized area after the data is sorted, such as by stock symbol and timestamp, and are good candidates for BLOCK_DICT. By contrast, long CHAR/VARCHAR columns are not good candidates for BLOCK_DICT encoding.

CHAR and VARCHAR columns that contain 0x00 or 0xFF characters should not be encoded with BLOCK_DICT. Also, BINARY/VARBINARY columns do not support BLOCK_DICT encoding.

BLOCK_DICT encoding requires significantly higher CPU usage than default encoding schemes. The maximum data expansion is eight percent (8%).

BLOCKDICT_COMP

This encoding type is similar to BLOCK_DICT except dictionary indexes are entropy coded. This encoding type requires significantly more CPU time to encode and decode and has a poorer worst-case performance. However, if the distribution of values is extremely skewed, using BLOCK_DICT_COMP encoding can lead to space savings.

BZIP_COMP

BZIP_COMP encoding uses the bzip2 compression algorithm on the block contents. See [bzip](#) web site for more information. This algorithm results in higher compression than the automatic LZO and gzip encoding; however, it requires more CPU time to compress. This algorithm is best

used on large string columns such as VARCHAR, VARBINARY, CHAR, and BINARY. Choose this encoding type when you are willing to trade slower load speeds for higher data compression.

COMMONDELTA_COMP

This compression scheme builds a dictionary of all deltas in the block and then stores indexes into the delta dictionary using entropy coding.

This scheme is ideal for sorted FLOAT and INTEGER-based (DATE/TIME/TIMESTAMP/INTERVAL) data columns with predictable sequences and only occasional sequence breaks, such as timestamps recorded at periodic intervals or primary keys. For example, the following sequence compresses well: 300, 600, 900, 1200, 1500, 600, 1200, 1800, 2400. The following sequence does not compress well: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55.

If delta distribution is excellent, columns can be stored in less than one bit per row. However, this scheme is very CPU intensive. If you use this scheme on data with arbitrary deltas, it can cause significant data expansion.

DELTARANGE_COMP

This compression scheme is primarily used for floating-point data; it stores each value as a delta from the previous one.

This scheme is ideal for many-valued FLOAT columns that are sorted or confined to a range. Do not use this scheme for unsorted columns that contain NULL values, as the storage cost for representing a NULL value is high. This scheme has a high cost for both compression and decompression.

To determine if DELTARANGE_COMP is suitable for a particular set of data, compare it to other schemes. Be sure to use the same sort order as the projection, and select sample data that will be stored consecutively in the database.

DELTAVAL

For INTEGER and DATE/TIME/TIMESTAMP/INTERVAL columns, data is recorded as a difference from the smallest value in the data block. This encoding has no effect on other data types.

DELTAVAL is best used for many-valued, unsorted integer or integer-based columns. CPU requirements for this encoding type are minimal, and data never expands.

GCDDELTA

For INTEGER and DATE/TIME/TIMESTAMP/INTERVAL columns, and NUMERIC columns with 18 or fewer digits, data is recorded as the difference from the smallest value in the data block divided by the greatest common divisor (GCD) of all entries in the block. This encoding has no effect on other data types.

ENCODING GCDDELTA is best used for many-valued, unsorted, integer columns or integer-based columns, when the values are a multiple of a common factor. For example, timestamps are stored internally in microseconds, so data that is only precise to the millisecond are all multiples of 1000. The CPU requirements for decoding GCDDELTA encoding are minimal, and the data never expands, but GCDDELTA may take more encoding time than DELTAVAL.

GZIP_COMP

This encoding type uses the gzip compression algorithm. See [gzip](#) web site for more information. This algorithm results in better compression than the automatic LZO compression, but lower compression than BZIP_COMP. It requires more CPU time to compress than LZO but less CPU time than BZIP_COMP. This algorithm is best used on large string columns such as VARCHAR, VARBINARY, CHAR, and BINARY. Use this encoding when you want a better compression than LZO, but at less CPU time than bzip2.

RLE

RLE (run length encoding) replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, it is best used for low cardinality columns that are present in the ORDER BY clause of a projection.

The Vertica execution engine processes RLE encoding run-by-run and the Vertica optimizer gives it preference. Use it only when run length is large, such as when low-cardinality columns are sorted.

The storage for RLE and AUTO encoding of CHAR/VARCHAR and BINARY/VARBINARY is always the same.

Valid Encoding for Numeric Data Types

- Valid encoding types for numeric data type columns with precision ≤ 18 include AUTO, BLOCK_DICT, BLOCKDICT_COMP, COMMONDELTA_COMP, DELTAVAL, GCDELTA, and RLE.
- Valid encoding types for numeric data type columns with precision > 18 include AUTO, BLOCK_DICT, BLOCKDICT_COMP, RLE.
- For information on numeric data types, see the section, [Numeric Data Types](#).

Hash Segmentation Clause

Specifies how to segment projection data for distribution across some or all cluster nodes. You can specify segmentation for a table and a projection. If a table definition specifies segmentation, Vertica uses it for that table's [auto-projections](#).

It is strongly recommended that you use Vertica's built-in [HASH](#) function, which distributes data evenly across the cluster, and facilitates optimal query execution.

Syntax

```
SEGMENTED BY expression { ALL NODES [ OFFSET offset ] | NODES node [ ,... ] }
```

Parameters

SEGMENTED BY <i>expression</i>	<p>A general SQL expression. Hash segmentation is the preferred method of segmentation. Vertica recommends using its built-in HASH function, whose arguments resolve to table columns. If you use an expression other than HASH, Vertica issues a warning.</p> <p>The segmentation expression should specify columns with a large number of unique data values and acceptable skew in their data distribution. In general, primary key columns that meet these criteria are good candidates for hash segmentation.</p> <p>For details, see Expression Requirements below.</p>
ALL NODES	Automatically distributes data evenly across all nodes when the

	projection is created. Node ordering is fixed.
OFFSET <i>offset</i>	A zero-based offset that indicates on which node to start segmentation distribution. This option is not valid for CREATE TABLE and CREATE TEMPORARY TABLE .
NODES <i>node</i> [, ...]	Specifies a subset of the nodes in the cluster over which to distribute projection data. You can specify a node only once. To obtain a list of all cluster nodes, query the system table V_CATALOG.NODES .

Expression Requirements

A segmentation expression must specify table columns as they are defined in the source table. Projection column names are not supported.

The following restrictions apply to segmentation expressions:

- All leaf expressions must be constants or [column references](#) to a column in the CREATE PROJECTION 's SELECT list.
- The expression must return the same value over the life of the database.
- Aggregate functions are not allowed.
- The expression must return non-negative INTEGER values in the range $0 \leq x < 2^{63}$, and values are generally distributed uniformly over that range.

Note: If the expression produces a value outside the expected range—for example, a negative value—no error occurs, and the row is added to the projection's first segment.

Examples

The following CREATE PROJECTION statement creates projection `public.employee_dimension_super`. It specifies to include all columns in table `public.employee_dimension`. The hash segmentation clause invokes the Vertica HASH function to segment projection data on the column `employee_key`; it also includes the ALL NODES clause, which specifies to distribute projection data evenly across all nodes in the cluster:

```
=> CREATE PROJECTION public.employee_dimension_super  
  AS SELECT * FROM public.employee_dimension  
  ORDER BY employee_key  
  SEGMENTED BY hash(employee_key) ALL NODES;
```

GROUPED Clause

Groups two or more columns into a single disk file. This minimizes file I/O for work loads that:

- Read a large percentage of the columns in a table.
- Perform single row look-ups.
- Query against many small columns.
- Frequently update data in these columns.

If you have data that is always accessed together and it is not used in predicates, you can increase query performance by grouping these columns. Once grouped, queries can no longer independently retrieve from disk all records for an individual column independent of the other columns within the group.

Note: RLE encoding is reduced when an RLE column is grouped with one or more non-RLE columns.

When grouping columns you can:

- Group some of the columns:
(a, GROUPED(b, c), d)
- Group all of the columns:
(GROUPED(a, b, c, d))
- Create multiple groupings in the same projection:
(GROUPED(a, b), GROUPED(c, d))

Note: Vertica performs dynamic column grouping. For example, to provide better read and write efficiency for small loads, Vertica ignores any projection-defined column grouping (or lack thereof) and groups all columns together by default.

CREATE PROJECTION (Live Aggregate Projections)

Creates metadata for live aggregate projections in the Vertica catalog. Top-K projections are a type of live aggregate projection.

Information here focuses on creating live aggregate projections. For details about creating other types of projections, including projections with expressions, see [CREATE PROJECTION](#).

Syntax

Grouping aggregate function results

```
CREATE PROJECTION [ IF NOT EXISTS ] projection-name
...[ (
.....{ projection-col | grouped-clause
..... [ ENCODING encoding-type ]
..... [ ACCESSRANK integer ]
.....} [,... ]
....) ]
AS SELECT {table-col | expr-with-table-cols } [,... ] FROM table-reference
... GROUP BY column-expr
... [ KSAFE [ k-num ] ]
```

Top-K aggregation

```
CREATE PROJECTION [ IF NOT EXISTS ] projection-name
...[ (
.....{ projection-col | grouped-clause
..... [ ENCODING encoding-type ]
..... [ ACCESSRANK integer ]
.....} [,... ]
.....)
... ]
AS SELECT {table-col | expr-with-table-cols } [ , ... ] FROM table-reference
... LIMIT num-rows OVER (PARTITION BY column-expr ORDER BY column-expr)
... [ KSAFE [ k-num ] ]
```

Parameters

IF NOT EXISTS	Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK
---------------	--

	<p>error message. In both cases, the object is not created.</p> <p>The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.</p> <p>For related information, see ON_ERROR_STOP.</p>
<i>projection-name</i>	<p>The name of the projection to create, where <i>projection-name</i> conforms to conventions described in Identifiers. Vertica creates the projection in the same schema as the anchor table.</p>
<i>projection-col</i>	<p>The name of a projection column.</p> <p>If you do not specify projection column names, Vertica uses the anchor table column names in the SELECT statement.</p>
<i>grouped-clause</i>	<p>See GROUPED Clause.</p>
ENCODING <i>encoding-type</i>	<p>Specifies the column encoding type, by default set to AUTO.</p>
ACCESSRANK <i>integer</i>	<p>Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking.</p>
<i>table-col</i> <i>expr-with-table-cols</i>	<p>A table column or expression of table columns to be included in the projection. If you specify projection column names, the two lists of projection columns and table columns/expressions must exactly match in number and order.</p>
<i>table-reference</i>	<p>A schema table with the columns to include in the projection, as follows:</p> <pre><i>table-name</i> [AS] <i>alias</i> [(<i>column-alias</i> [,...])]</pre>
GROUP BY <i>column-expr</i> [,...]	<p>One or more column expressions from the SELECT list. The first <i>column-expr</i> must be the first column expression in the SELECT list, the second <i>column-expr</i> must be the second column expression in the SELECT list, and so on.</p>
LIMIT <i>num-rows</i>	<p>The number of rows to return from the specified partition.</p>

<p>OVER (PARTITION BY <i>column-expr</i> [, ...]</p>	<p>Specifies window partitioning by one or more column expressions from the SELECT list. The first <i>column-expr</i> is the first column expression in the SELECT list, the second <i>column-expr</i> is the second column expression in the SELECT list, and so on.</p>
<p>ORDER BY <i>column-expr</i> [, ...] [ASC DESC]</p>	<p>The order in which the top <i>k</i> rows are returned, by default in ascending (ASC) order. All column expressions must be from the SELECT list, where the first <i>column-expr</i> must be the first column expression in the SELECT list to follow the last PARTITION BY column expression.</p> <p>Top-K projections support ORDER BY NULLS FIRST/LAST.</p>
<p>KSAFE [<i>k-num</i>]</p>	<p>Specifies K-safety for the projection, where <i>k-num</i> must be equal to or greater than system K-safety. Vertica ignores this parameter if set for unsegmented projections. If you omit <i>k-num</i>, Vertica uses system K-safety.</p> <p>Vertica sets projection K-safety as follows:</p> <ul style="list-style-type: none"> • KSAFE and OFFSET clause omitted: Uses system K-safety. • OFFSET clause is omitted and KSAFE is specified: Uses KSAFE setting. • KSAFE is omitted and the OFFSET clause is specified: Uses OFFSET setting. <p>If the CREATE PROJECTION statement specifies KSAFE and the OFFSET clause, Vertica returns an error.</p> <p>For general information, see K-Safety in Vertica Concepts.</p>

Privileges

See [Projection Privileges](#)

Requirements and Restrictions

See:

- [Creating Live Aggregate Projections](#)
- [Creating Top-K Projections](#)

Examples

See:

- [Live Aggregate Projection Example](#)
- [Top-K Projection Examples](#)

CREATE PROJECTION (UDTFs)

Creates metadata in the Vertica catalog for projections that invoke user-defined transform functions (UDTFs).

Important: Currently, live aggregate projections can only reference UDTFs that are developed in C++.

Syntax

```
CREATE PROJECTION [ IF NOT EXISTS ] projection-name
...[ (
.....{ projection-col | grouped-clause
..... [ ENCODING encoding-type ]
..... [ ACCESSRANK integer ]
.....} [,... ]
...) ]
AS {
.....batch-query FROM { prepass-query sq-results | table-ref }
.....| prepass-query
...}
```

batch-query

```
SELECT { table-column | expr-with-table-cols }[,...], batch-udtf(batch-args)
...OVER (PARTITION BATCH BY partition-col-expr[,...] )
...[ AS (batch-output-cols) ]
```

prepass-query

```
SELECT { table-col | expr-with-table-cols }[,...], prepass-udtf(prepass-args)
...OVER (PARTITION PREPASS BY partition-col-expr[,...] )
...[ AS (prepass-output-cols) ] FROM table-ref
```

Parameters

<p>IF NOT EXISTS</p>	<p>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.</p> <p>The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.</p> <p>For related information, see ON_ERROR_STOP.</p>
<p><i>projection-name</i></p>	<p>The name of the projection to create, where <i>projection-name</i> conforms to conventions described in Identifiers. Vertica creates the projection in the same schema as the anchor table.</p>
<p><i>projection-column</i></p>	<p>The name of a projection column.</p> <p>If you do not specify projection column names, Vertica uses the anchor table column names that are specified in the SELECT statement.</p>
<p><i>grouped-clause</i></p>	<p>See GROUPED Clause.</p>
<p>ENCODING <i>encoding-type</i></p>	<p>Specifies the column encoding type, by default set to AUTO.</p>
<p>ACCESSRANK <i>integer</i></p>	<p>Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking.</p>
<p><i>table-col</i> <i>expr-with-table-cols</i></p>	<p>A table column or expression of table columns to include in the projection.</p>
<p><i>batch-udtf(batch-args)</i></p>	<p>The batch UDTF to invoke each time the following events occur:</p> <ul style="list-style-type: none"> • Tuple mover mergeout

	<ul style="list-style-type: none"> • Queries on the projection • If invoked singly, on data load operations <p>Important: If the projection definition includes a pre-pass subquery, <i>batch-args</i> must exactly match the pre-pass UDTF output columns, in name and order.</p>
<i>prepass-udtf</i> (<i>prepass-args</i>)	<p>The pre-pass UDTF to invoke on each load operation such as COPY or INSERT.</p> <p>If specified in a subquery, the pre-pass UDTF returns transformed data to the batch query for further processing. Otherwise, the pre-pass query results are added to projection data storage.</p>
OVER (PARTITION BATCH BY <i>partition-col-expr</i> [, ...])	<p>Specifies the UDTF type and how to partition the data it returns:</p> <ul style="list-style-type: none"> • BATCH identifies the UDTF as a batch UDTF. • PREPASS identifies the UDTF as a pre-pass UDTF. <p>In both cases, the OVER clause specifies partitioning with one or more column expressions from the SELECT list. The first <i>partition-col-expr</i> is the first column expression in the SELECT list, the second <i>partition-col-expr</i> is the second column expression in the SELECT list, and so on.</p> <p>Note: The projection is implicitly segmented and ordered on PARTITION BY columns.</p>
AS (<i>batch-output-cols</i>) AS (<i>prepass-output-cols</i>)	<p>Optionally names columns that are returned by the UDTF.</p> <p>If a pre-pass subquery omits this clause, the outer batch query UDTF arguments (<i>batch-args</i>) must reference the column names as they are defined in the pre-pass UDTF.</p>
<i>table-ref</i>	<p>A schema table with the columns to include in the projection.</p>
<i>sq-results</i>	<p>Subquery result set that is returned to the outer batch UDTF.</p>

Privileges

See [Projection Privileges](#). The projection creator also must have EXECUTE privileges on all UDTFs that are referenced by the projection.

Restrictions

Vertica does not regard live aggregate projections as superprojections, even one that includes all table columns.

UDTF Types

CREATE PROJECTION can define live aggregate projections that invoke user-defined transform functions (UDTFs). Vertica invokes UDTFs at multiple points of projection processing:

- **Pre-pass UDTFs:** Invoked when data is loaded into the projection's anchor table—for example through COPY or INSERT statements. A pre-pass UDTF transforms the new data before it is stored in the projection's ROS containers. You identify a pre-pass UDTF in the projection's PARTITION BY clause, through the keyword PREPASS.
- **Batch UDTFs:** Invoked on three events: the Vertica tuple mover consolidates stored projection data (mergeout); the projection is queried; and if invoked singly, data load operations. In all cases, the UDTF aggregates projection data and stores the aggregated results. Aggregation is cumulative across mergeout and load operations, and is completed (if necessary) on query execution. You identify a batch UDTF in the projection's PARTITION BY clause, through the keyword BATCH.

Vertica stores all UDTF results in projection ROS containers, thereby enabling faster response time when you query the projection.

UDTF Specification Options

A projection definition can specify up to two UDTFs, in any of the following ways:

- **Single pre-pass UDTF:** The pre-pass UDTF transforms newly loaded data and stores it in the projection. Use the following syntax:

```
=> CREATE PROJECTION projection-name AS SELECT ..., udtf(args)  
    OVER(PARTITION PREPASS BY partition-cols) AS (prepass-output-columns) FROM table-ref;
```

- **Single batch UDTF:** The batch UDTF transforms and aggregates projection data on mergeout, insert, and query operations. Use the following syntax:

```
=> CREATE PROJECTION projection-name AS SELECT ..., udtf(args)  
    OVER(PARTITION BATCH BY partition-cols) AS (batch-output-columns) FROM table-ref;
```

- **Pre-pass and batch UDTFs:** You can define a projection with a subquery that invokes a pre-pass UDTF. The pre-pass UDTF returns transformed data to the top-level batch query. The batch UDTF then iteratively aggregates all projection data.

Use the following syntax:

```
=> CREATE PROJECTION projection-name AS SELECT ..., batch-udtf(batch-args)  
    OVER ( PARTITION BATCH BY partition-cols ) AS (batch-output-columns)  
    FROM ( SELECT ..., prepass-udtf(prepass-args)  
          OVER ( PARTITION PREPASS BY partition-cols) AS (prepass-output-columns)  
          FROM table-ref ) sq-ref;
```

Examples

See [Examples](#) in [Pre-Aggregating UDTF Results](#).

See Also

[Pre-Aggregating UDTF Results](#)

CREATE RESOURCE POOL

Creates a custom resource pool and sets one or more resource pool parameters.

Syntax

```
CREATE RESOURCE POOL pool-name [ parameter-name setting ]...
```

Parameters

Note: You can set all resource pool parameters to their DEFAULT value. The [V_CATALOG.RESOURCE_POOL_DEFAULTS](#) system table contains default parameter values. Query this table to determine default settings for all resource pools.

Default values specified in this table pertain only to user-defined resource pools. For built-in pool default values, see [Built-In Pool Configuration](#).

<i>pool-name</i>	The name of the resource pool. Built-in pool names cannot be used for user-defined pools.
<i>parameter-name</i>	The parameter to set, listed below.
CASCADE TO	Specifies a secondary resource pool for executing queries that exceed the RUNTIMECAP setting of their assigned resource pool: <code>CASCADE TO <i>secondary-pool</i></code>
CPUAFFINITYMODE	Specifies whether the resource pool has exclusive or shared use of the CPUs specified in CPUAFFINITYSET : <pre>CPUAFFINITYMODE { SHARED EXCLUSIVE ANY }</pre> <ul style="list-style-type: none"> • SHARED: Queries that run in this pool share its CPUAFFINITYSET CPUs with other Vertica resource pools. • EXCLUSIVE: Dedicates CPUAFFINITYSET CPUs to this resource pool only, and excludes other Vertica resource pools. If CPUAFFINITYSET is set as a percentage, then that percentage of CPU resources available to Vertica is assigned solely for this resource pool. • ANY (default): Queries in this resource pool can run on any CPU, invalid if CPUAFFINITYSET designates CPU resources.

<p>CPUAFFINITYSET</p>	<p>Specifies which CPUs are available to this resource pool. All cluster nodes must have the same number of CPUs. The CPU resources assigned to this set are unavailable to general resource pools.</p> <pre>CPUAFFINITYSET { 'cpu-index[,...]' 'cpu-index_i-cpu-index_n' 'integer%' NONE }</pre> <ul style="list-style-type: none"> • <i>cpu-index[,...]</i>: Dedicates one or more comma-delimited CPUs to this pool. • <i>cpu-index_i-cpu-index_n</i>: Dedicates a range of contiguous CPU indexes to this pool • <i>integer%</i>: Percentage of all available CPUs to use for this pool. Vertica rounds this percentage down to include whole CPU units. • NONE (default): No affinity set is assigned to this resource pool. The queries associated with this pool are executed on any CPU.
<p>EXECUTIONPARALLELISM</p>	<p>Limits the number of threads used to process any single query issued in this resource pool.</p> <pre>EXECUTIONPARALLELISM { integer AUTO }</pre> <ul style="list-style-type: none"> • <i>integer</i>: A value between 1 and the number of cores. Setting this parameter to a reduced value increases throughput of short queries issued in the pool, especially if the queries are executed concurrently. • AUTO (default): Vertica sets this value based on the number of cores, available memory, and amount of data in the system. Unless memory is limited, or the amount of data is very small, Vertica sets this value to the number of cores on the node.
<p>MAXCONCURRENCY</p>	<p>Sets the maximum number of concurrent execution slots available to the resource pool, across the cluster:</p> <pre>MAXCONCURRENCY { integer NONE }</pre>

	<p>NONE (default) specifies unlimited number of concurrent execution slots.</p>
<p>MAXMEMORYSIZE</p>	<p>The maximum size per node the resource pool can grow by borrowing memory from the GENERAL pool:</p> <pre>MAXMEMORYSIZE { 'integer%' 'integer{K M G T}' NONE }</pre> <ul style="list-style-type: none"> • <i>integer%</i>: Percentage of total memory • <i>integer{K M G T}</i>: Amount of memory in kilobytes, megabytes, gigabytes, or terabytes • NONE: Unlimited; pool can borrow any amount of available memory from the GENERAL pool.
<p>MEMORYSIZE</p>	<p>The amount of total memory available to the Vertica resource manager that is allocated to this pool per node:</p> <pre>MEMORYSIZE { 'integer%' 'integer{K M G T}' }</pre> <ul style="list-style-type: none"> • <i>integer%</i>: Percentage of total memory • <i>integer{K M G T}</i>: Amount of memory in kilobytes, megabytes, gigabytes, or terabytes <p>Default: 0%. No memory allocated, the resource pool borrows memory from the GENERAL pool.</p>
<p>PLANNEDCONCURRENCY</p>	<p>Specifies the preferred number queries to execute concurrently in the resource pool. This setting applies to the entire cluster:</p> <pre>PLANNEDCONCURRENCY { integer AUTO }</pre> <ul style="list-style-type: none"> • <i>integer</i>: The preferred number of concurrently executing queries. When possible, query resource budgets are limited to allow this level of concurrent execution. • AUTO (default): Value is calculated automatically at

	<p>query runtime. Vertica sets this parameter to the lower of these two calculations, but never less than 4:</p> <ul style="list-style-type: none"> ■ Number of logical cores ■ Memory divided by 2GB <p>For clusters where the number of logical cores differs on different nodes, AUTO can apply differently on each node. Distributed queries run like the minimal effective planned concurrency. Single node queries run with the planned concurrency of the initiator.</p> <p>Tip: Change this parameter only after evaluating performance over a period of time.</p>
<p>PRIORITY</p>	<p>Specifies priority of queries in this pool when they compete for resources in the GENERAL pool:</p> <p>PRIORITY { <i>integer</i> HOLD }</p> <ul style="list-style-type: none"> • <i>integer</i>: A negative or positive integer value, where higher numbers denote higher priority: <ul style="list-style-type: none"> ■ User-defined pools: -100 to 100 ■ Built-in pools SYSQUERY, RECOVERY, and TM: -110 to 110 • HOLD: Sets priority to -999. Queries in this pool are queued until QUEUE_TIMEOUT is reached. <p>Default: 0</p>
<p>QUEUE_TIMEOUT</p>	<p>Specifies how long a request can wait for pool resources before it is rejected:</p> <p>QUEUE_TIMEOUT { <i>integer</i> NONE }</p> <ul style="list-style-type: none"> • <i>integer</i>: Maximum wait time in seconds • NONE: No maximum wait time, request can be queued indefinitely. <p>Default: 300 seconds</p>
<p>RUNTIMECAP</p>	<p>Prevents runaway queries by setting the maximum time</p>

	<p>a query in the pool can execute. If a query exceeds this setting, it tries to cascade to a secondary pool:</p> <p><code>RUNTIMECAP { <i>interval</i> NONE }</code></p> <ul style="list-style-type: none"> • <i>interval</i>: An interval of 1 minute or 100 seconds; should not exceed one year. • NONE: No time limit on queries running in this pool. <p>To specify a value in days, provide an integer value. To provide a value less than one day, provide the interval in the format <code>hours:minutes:seconds</code>. For example a value of <code>1:30:00</code> would equal 90 minutes.</p> <p>If the user or session also has a RUNTIMECAP, the shorter limit applies.</p>
<p>RUNTIMEPRIORITY</p>	<p>Determines how the resource manager should prioritize dedication of run-time resources (CPU, I/O bandwidth) to queries already running in this resource pool:</p> <p><code>RUNTIMEPRIORITY { HIGH MEDIUM LOW }</code></p> <p>Default: MEDIUM</p>
<p>RUNTIMEPRIORITYTHRESHOLD</p>	<p>Specifies in seconds a time limit in which a query must finish before the resource manager assigns to it the resource pool's RUNTIMEPRIORITY. All queries begin running at a HIGH priority. When a query's duration exceeds this threshold, it is assigned the RUNTIMEPRIORITY of the resource pool.</p> <p><code>RUNTIMEPRIORITYTHRESHOLD <i>seconds</i></code></p> <p>Default: 2</p>

Privileges

Superuser

Examples

This example shows how to create a resource pool with MEMORYSIZE of 1800 MB.

```
=> CREATE RESOURCE POOL ceo_pool MEMORYSIZE '1800M' PRIORITY 10;  
CREATE RESOURCE POOL
```

Assuming the CEO report user already exists, associate this user with the preceding resource pool using ALTER USER statement.

```
=> GRANT USAGE ON RESOURCE POOL ceo_pool to ceo_user;  
GRANT PRIVILEGE  
=> ALTER USER ceo_user RESOURCE POOL ceo_pool;  
ALTER USER
```

Issue the following command to confirm that the ceo_user is associated with the ceo_pool:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';  
-[ RECORD 1 ]-----+-----  
user_id           | 45035996273733402  
user_name         | ceo_user  
is_super_user     | f  
profile_name      | default  
is_locked         | f  
lock_time         |  
resource_pool     | ceo_pool  
memory_cap_kb     | unlimited  
temp_space_cap_kb | unlimited  
run_time_cap      | unlimited  
all_roles         |  
default_roles     |  
search_path       | "$user", public, v_catalog, v_monitor, v_internal
```

This example shows how to create and designate secondary resource pools.

```
=> CREATE RESOURCE POOL rp3 RUNTIMECAP '5 minutes';  
=> CREATE RESOURCE POOL rp2 RUNTIMECAP '3 minutes' CASCADE TO rp3;  
=> CREATE RESOURCE POOL rp1 RUNTIMECAP '1 minute' CASCADE TO rp2;  
=> SET SESSION RESOURCE_POOL = rp1;
```

See Also

- [ALTER RESOURCE POOL](#)
- [CREATE USER](#)
- [DROP RESOURCE POOL](#)
- [SET SESSION RESOURCE_POOL](#)
- [SET SESSION MEMORYCAP](#)
- [Managing Workloads](#)

Built-In Pools

Vertica is preconfigured with built-in pools for various system tasks:

- [GENERAL](#)
- [BLOBDATA](#)
- [DBD](#)
- [JVM](#)
- [METADATA](#)
- [RECOVERY](#)
- [REFRESH](#)
- [SYSDATA](#)
- [SYSQUERY](#)
- [TM](#)
- [WOSDATA](#)

Built-in pools can be customized to suit your usage requirements. See [ALTER RESOURCE POOL](#) for details on resource pool settings.

GENERAL

A special, catch-all pool used to answer requests that have no specific resource pool associated with them. Any memory left over after memory has been allocated to all other pools is automatically allocated to the GENERAL pool. The [MEMORYSIZE](#) parameter of the GENERAL pool is undefined (variable), however, the GENERAL pool must be at least 1GB in size and cannot be smaller than 25% of the memory in the system.

The [MAXMEMORYSIZE](#) parameter of the GENERAL pool has special meaning; when set as a % value it represents the percent of total physical RAM on the machine that the Resource Manager can use for queries. By default, it is set to 95%. The `GENERAL . MAXMEMORYSIZE` governs the total amount of RAM that the Resource Manager can use for queries, regardless of whether it is set to a percent or to a specific value (for example, '10GB')

User-defined pools can borrow memory from the GENERAL pool to satisfy requests that need extra memory until the MAXMEMORYSIZE parameter of that pool is reached. If the pool is configured to have MEMORYSIZE equal to MAXMEMORYSIZE, it cannot borrow any memory from the GENERAL pool and is said to be a standalone resource pool. When multiple pools request memory from the GENERAL pool, they are granted access to general pool memory according to their priority setting. In this manner, the GENERAL pool provides some elasticity to account for point-in-time deviations from normal usage of individual resource pools.

Vertica recommends reducing the GENERAL pool MAXMEMORYSIZE if your catalog uses over 5% of overall memory.

BLOBDATA

The BLOBDATA pool controls resource usage for in-memory blobs. *In-memory blobs* are objects used by a number of the machine learning SQL functions. You should adjust this pool if you plan on processing large machine learning workloads. For information about tuning the pool, see [Tuning for Machine Learning](#).

If a query using the BLOBDATA pool exceeds its query planning budget, then it spills to disk. For more information about tuning your query budget, see [Target Memory Determination for Queries in Concurrent Environments](#).

DBD

The DBD pool controls resource usage for Database Designer processing. Use of this pool is enabled by configuration parameter `DBDUseOnlyDesignerResourcePool`, by default set to false.

By default, `QUEUETIMEOUT` is set to 0 for this pool. When resources are under pressure, this setting causes the DBD to time out immediately, and not be queued to run later. Database Designer then requests the user to run the designer later, when resources are more available.

Important: Do not change `QUEUETIMEOUT` or other DBD pool settings.

JVM

The JVM pool controls Java Virtual Machine resources used by Java User Defined Extensions. When a Java UDX starts the JVM, it draws resources from the those specified in the JVM resource pool. Vertica does not reserve memory in advance for the JVM pool. When needed, the pool can expand to 10% of physical memory or 2 GB of memory, whichever is smaller. If

you are buffering large amounts of data, you may need to increase the size of the JVM resource pool.

You can adjust the size of your JVM resource pool by changing its configuration settings. Unlike other resource pools, the JVM resource pool does not release resources until a session is closed.

METADATA

The pool that tracks memory allocated for catalog data and storage data structures. This pool increases in size as Vertica metadata consumes additional resources. Memory assigned to the METADATA pool is subtracted from the GENERAL pool, enabling the Vertica resource manager to make more effective use of available resources. If the METADATA resource pool reaches 75% of the GENERAL pool, Vertica stops updating METADATA memory size and displays a warning message in the the vertica.log file. You can enable or disable the METADATA pool with the EnableMetadataMemoryTracking general parameter.

If you have created a "dummy" or "swap" resource pool to protect resources for use by your operating system, you can replace that pool with the METADATA pool.

Users cannot change the parameters of the METADATA resource pool.

RECOVERY

The pool used by queries issued when recovering another node of the database. The [MAXCONCURRENCY](#) parameter is used to determine how many concurrent recovery threads to use. You can use the [PLANNEDCONCURRENCY](#) parameter (by default, set to twice the MAXCONCURRENCY) to tune how to apportion memory to recovery queries.

See [Tuning for Recovery](#) in the Administrator's Guide.

REFRESH

The pool used by queries issued by [PROJECTION_REFRESHES](#) operations. Refresh does not currently use multiple concurrent threads; thus, changes to the [MAXCONCURRENCY](#) values have no effect.

See [Scenario: Tuning for Refresh](#) in the Administrator's Guide.

SYSDATA

The pool reserved for temporary storage of intermediate results of queries against [system monitoring and catalog tables](#). If the SYSDATA pool size is too low, Vertica cannot execute queries for large system tables or during high concurrent access to system tables.

Note: [MAXMEMORYSIZE](#) of the SYSDATA pool cannot be changed if any of its memory is in use.

SYSQUERY

The pool that runs queries against [system monitoring and catalog tables](#). The SYSQUERY pool reserves resources for system table queries so that they are never blocked by contention for available resources.

TM

The Tuple Mover (TM) pool. You can set the [MAXCONCURRENCY](#) parameter for the TM pool to allow more than one concurrent TM operation to occur.

See [Tuning Tuple Mover Pool Settings](#) in the Administrator's Guide.

WOSDATA

The Write Optimized Store (WOS) resource pool. Data loads to the WOS automatically spill to the ROS once it exceeds a certain amount of WOS usage; the [PLANNEDCONCURRENCY](#) parameter of the WOS is used to determine this spill threshold. For instance, if [PLANNEDCONCURRENCY](#) of the WOSDATA pool is set to 4, once a load has occupied one quarter of the WOS, it spills to the ROS.

The WOSDATA pool is limited to a size of 2GB or 25% of the GENERAL pool's limits, whichever is less. However, when the GENERAL pool has a limit of 20GB or higher, the WOSDATA pool will instead have 2GB of dedicated memory.

Note: [MAXMEMORYSIZE](#) of the WOSDATA pool cannot be changed if any of its memory is in use. For example, you cannot change [MAXMEMORYSIZE](#) unless you first disable trickle loading jobs and wait until the WOS is empty.

See [Scenario: Tuning for Continuous Load and Query](#) in the Administrator's Guide.

Built-In Pool Configuration

The tables in this section list the default configuration settings for the Vertica built-in resource pools:

- [GENERAL](#)
- [BLOBDATA](#)
- [DBD](#)
- [JVM](#)
- [METADATA](#)
- [RECOVERY](#)
- [REFRESH](#)
- [SYSDATA](#)
- [SYSQUERY](#)
- [TM](#)
- [WOSDATA](#)

Some built-in resource pool parameter values have restrictions, which are noted in the tables. For guidance about how to tune the General resource pool, see [Best Practices for Managing Workload Resources](#) in the Administrator's Guide.

GENERAL

Setting	Value
MEMORYSIZE	N/A (cannot be set)
MAXMEMORYSIZE	Sets the maximum memory to use for all resource pools. Specify this value as a percentage of total RAM. For example, if your node has 64GB of memory, setting MAXMEMORYSIZE to 50% allocates half of the available memory. The total memory for all resource pools

Setting	Value
	<p>cannot exceed 32GB.</p> <p>Setting this parameter to 100% generates warning messages that swapping could result. Changing the MAXMEMORYSIZE parameter has the following restrictions:</p> <ul style="list-style-type: none"> • Must be 1GB or greater. • Cannot be less than 25% of total system RAM. <p>Default: 95%, with minimum of 25% (at least 1GB), maximum 100%</p>
PRIORITY	0
RUNTIMEPRIORITY	Medium
RUNTIMEPRIORITYTHRESHOLD	2
QUEUE_TIMEOUT	300
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	<p>An integer representing the number of concurrent queries you expect to run against the resource pool. When set to the default value of AUTO, Vertica automatically sets PLANNEDCONCURRENCY at query runtime, choosing the lower of these two values:</p> <ul style="list-style-type: none"> • Number of cores • Memory/2GB <p>The value 4 is the minimum value for PLANNEDCONCURRENCY.</p> <p>For systems with a large number of cores, the PLANNEDCONCURRENCY setting for the GENERAL pool defaults to a value that is too low. For this configuration, Vertica recommends adjusting to a value equaling the number of cores:</p> <pre>ALTER RESOURCE POOL general PLANNEDCONCURRENCY <#cores>;</pre>

Setting	Value
	Default: AUTO
MAXCONCURRENCY	Unlimited Restrictions: Setting to 0 generates warnings that no system queries may be able to run in the system.
SINGLEINITIATOR	False. This parameter is included for backwards compatibility only. Do not change the value.

BLOBDATA

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited
EXECUTIONPARALLELISM	N/A (cannot be set)
PRIORITY	N/A (cannot be set)
RUNTIMEPRIORITY	N/A (cannot be set)
RUNTIMEPRIORITYTHRESHOLD	N/A (cannot be set)
QUEUETIMEOUT	N/A (cannot be set)
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	2
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	N/A (cannot be set)

DBD

Setting	Value
MEMORYSIZE	0%

Setting	Value
MAXMEMORYSIZE	Unlimited
EXECUTIONPARALLELISM	AUTO
PRIORITY	0
RUNTIMEPRIORITY	MEDIUM
RUNTIMEPRIORITYTHRESHOLD	0
QUEUE_TIMEOUT	0
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	See GENERAL
MAXCONCURRENCY	Unlimited
SINGLEINITIATOR	False. This parameter is included for backwards compatibility. Do not change the value.

JVM

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	10% of memory or 2 GB, whichever is smaller
EXECUTIONPARALLELISM	AUTO
PRIORITY	0
RUNTIMEPRIORITY	MEDIUM
RUNTIMEPRIORITYTHRESHOLD	2
QUEUE_TIMEOUT	300
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	AUTO

Setting	Value
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	FALSE. This parameter is included for backwards compatibility. Do not change the value.

METADATA

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited
EXECUTIONPARALLELISM	N/A (cannot be set)
PRIORITY	N/A (cannot be set)
RUNTIMEPRIORITY	N/A (cannot be set)
RUNTIMEPRIORITYTHRESHOLD	N/A (cannot be set)
QUEUE_TIMEOUT	N/A (cannot be set)
PLANNEDCONCURRENCY	N/A (cannot be set)
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	FALSE. This parameter is included for backwards compatibility. Do not change the value.
CPUAFFINITYSET	N/A (cannot be set)
CPUAFFINITYMODE	N/A (cannot be set)

RECOVERY

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited

Setting	Value
	Restrictions: cannot set to < 25%.
EXECUTIONPARALLELISM	AUTO
PRIORITY	107
RUNTIMEPRIORITY	MEDIUM
RUNTIMEPRIORITYTHRESHOLD	60
QUEUE_TIMEOUT	300
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	Twice MAXCONCURRENCY
MAXCONCURRENCY	(# of cores / 2) + 1 Restrictions: Cannot set to 0 or NONE (unlimited)
SINGLEINITIATOR	True. This parameter is included for backwards compatibility. Do not change the value.

REFRESH

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	Unlimited
EXECUTIONPARALLELISM	AUTO
PRIORITY	-10
RUNTIMEPRIORITY	MEDIUM
RUNTIMEPRIORITYTHRESHOLD	60
QUEUE_TIMEOUT	300
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	4

Setting	Value
	<p>Note: PLANNEDCONCURRENCY for the REFRESH pool is always set to 4 by default (the AUTO setting). For some of the resource pools other than REFRESH, PLANNEDCONCURRENCY is set according to a formula that considers your hardware configuration. PLANNEDCONCURRENCY for REFRESH does not consider hardware as the default setting is always 4. You can change the default setting of PLANNEDCONCURRENCY with the ALTER RESOURCE POOL statement.</p>
MAXCONCURRENCY	<p>Unlimited</p> <p>Restrictions: cannot set to 0</p>
SINGLEINITIATOR	<p>True. This parameter is included for backwards compatibility. Do not change the value.</p>

SYSDATA

Setting	Value
MEMORYSIZE	100m
MAXMEMORYSIZE	<p>10%</p> <p>Restriction: Setting To <4m generates warnings that no system queries may be able to run in the system.</p>
EXECUTIONPARALLELISM	N/A (cannot be set)
PRIORITY	N/A (cannot be set)
RUNTIMEPRIORITY	N/A (cannot be set)
RUNTIMEPRIORITYTHRESHOLD	N/A (cannot be set)
QUEUETIMEOUT	N/A (cannot be set)
RUNTIMECAP	N/A (cannot be set)
PLANNEDCONCURRENCY	N/A (cannot be set)

Setting	Value
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	N/A (cannot be set)

SYSQUERY

Setting	Value
MEMORYSIZE	64M Restrictions: Setting to <20M generates warnings because it could prevent system queries from running and make problem diagnosis difficult.
MAXMEMORYSIZE	Unlimited
EXECUTIONPARALLELISM	AUTO
PRIORITY	110
RUNTIMEPRIORITY	HIGH
RUNTIMEPRIORITYTHRESHOLD	0
QUEUE_TIMEOUT	300
RUNTIMECAP	NONE.
PLANNEDCONCURRENCY	See GENERAL
MAXCONCURRENCY	Unlimited Restrictions: Setting to 0 generates warnings that no system queries may be able to run in the system.
SINGLEINITIATOR	False. This parameter is included for backwards compatibility only. Do not change the value.

TM

Setting	Value
MEMORYSIZE	100M
MAXMEMORYSIZE	Unlimited
EXECUTIONPARALLELISM	AUTO
PRIORITY	105
RUNTIMEPRIORITY	MEDIUM
RUNTIMEPRIORITYTHRESHOLD	60
QUEUE_TIMEOUT	300
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	1
MAXCONCURRENCY	2 Restrictions: Cannot set to 0 or NONE (unlimited)
SINGLEINITIATOR	True. This parameter is included for backwards compatibility. Do not change the value.

WOSDATA

Setting	Value
MEMORYSIZE	0%
MAXMEMORYSIZE	25% or 2GB of the GENERAL pool's limits, whichever is less. When the GENERAL pool's memory limits are 20GB or greater, the WOSDATA pool will instead have a dedicated 2GB of memory.
EXECUTIONPARALLELISM	N/A (cannot be set)

Setting	Value
PRIORITY	N/A (cannot be set)
RUNTIMEPRIORITY	N/A (cannot be set)
RUNTIMEPRIORITYTHRESHOLD	N/A (cannot be set)
QUEUE_TIMEOUT	N/A (cannot be set)
RUNTIMECAP	NONE
PLANNEDCONCURRENCY	2
MAXCONCURRENCY	N/A (cannot be set)
SINGLEINITIATOR	N/A (cannot be set)

CREATE ROLE

Creates a new, empty role. You must then add permissions to the role using one of the GRANT statements.

Syntax

```
CREATE ROLE role;
```

Parameters

<i>role</i>	The name for the new role, where <i>role</i> conforms to conventions described in Identifiers .
-------------	---

Privileges

Must be a superuser to create a role.

Examples

This example shows to create an empty role called roleA.

```
=> CREATE ROLE roleA;  
CREATE ROLE
```

See Also

- [ALTER ROLE RENAME](#)
- [DROP ROLE](#)

CREATE SCHEMA

Defines a schema.

Syntax

```
CREATE SCHEMA [ IF NOT EXISTS ] schema  
... [ AUTHORIZATION username ]  
... [ DEFAULT { INCLUDE | EXCLUDE } [ SCHEMA ] PRIVILEGES ]
```

Parameters

IF NOT EXISTS	<p>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.</p> <p>The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.</p> <p>For related information, see ON_ERROR_STOP.</p>
---------------	--

<i>schema</i>	Specifies the name of the schema to create, where <i>schema</i> conforms to conventions described in Identifiers .
AUTHORIZATION <i>username</i>	Assigns ownership of the schema to a user. If a user name is not provided, the user who creates the schema is assigned ownership. Only superusers can create a schema that is owned by another user.
DEFAULT {INCLUDE EXCLUDE} [SCHEMA] PRIVILEGES	<p>Specifies whether to enable or disable inheritance of privileges for tables in the schema. INCLUDE PRIVILEGES grants the tables in the schema the same privileges granted to the schema.</p> <p>If you omit INCLUDE PRIVILEGES, you must grant privileges individually for each table in the schema.</p> <p>For more information see Grant Inherited Privileges.</p>

Privileges

- Superuser
- [CREATE privilege for the database](#)

Optionally, CREATE SCHEMA can include the following sub-statements to create tables within the schema:

- [CREATE TABLE](#)
- [GRANT Statements](#)

These sub-statements are treated as if they were entered as individual commands after CREATE SCHEMA executes. The following exceptions apply:

- The AUTHORIZATION statement indicates all tables are owned by the specified user.
- CREATE SCHEMA statement and all associated sub-statements are treated as a single transaction. If any statement fails, Vertica rolls back the entire CREATE SCHEMA statement.

Examples

The following example creates a schema named s1 with no objects.

```
=> CREATE SCHEMA s1;
```

The following command creates schema s2 if it does not already exist:

```
=> CREATE SCHEMA IF NOT EXISTS schema2;
```

If the schema already exists, Vertica returns a rollback message:

```
=> CREATE SCHEMA IF NOT EXISTS schema2;  
NOTICE 4214: Object "schema2" already exists; nothing was done
```

The following series of commands create a schema named s1 with a table named t1 and grants Fred and Aniket access to all existing tables and ALL privileges on table t1:

```
=> CREATE SCHEMA s1;  
=> CREATE TABLE t1 (c INT);  
=> GRANT USAGE ON SCHEMA s1 TO Fred, Aniket;  
=> GRANT ALL ON TABLE t1 TO Fred, Aniket;
```

This example sets the default behavior for new table t2 to automatically inherit the schema's privileges:

```
=> CREATE SCHEMA s1 DEFAULT INCLUDE SCHEMA PRIVILEGES;  
=> CREATE TABLE s1.t2(i int);
```

This example sets the default for new tables to not automatically inherit privileges from the schema:

```
=> CREATE SCHEMA s2 DEFAULT EXCLUDE SCHEMA PRIVILEGES;
```

See Also

- [ALTER SCHEMA](#)
- [SET SEARCH_PATH](#)
- [DROP SCHEMA](#)

CREATE SEQUENCE

Defines a new named sequence number generator object.

Use sequences or auto-incrementing columns for primary key columns. For example, to generate only even numbers in a sequence, specify a start value of 2, and increment the sequence by 2. Sequences guarantee uniqueness and avoid constraint enforcement problems and their associated overhead.

Syntax

```
CREATE SEQUENCE [schema.]sequence-name
... [ INCREMENT [ BY ] positive-or-negative ]
... [ MINVALUE minvalue | NO MINVALUE ]
... [ MAXVALUE maxvalue | NO MAXVALUE ]
... [ START [ WITH ] start ]
... [ CACHE value | NO CACHE ]
... [ CYCLE | NO CYCLE ]
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>sequence-name</i>	<p>The name of the sequence to create, where <i>sequence-name</i> conforms to conventions described in Identifiers. It must also be unique among sequences, tables, projections, and views.</p>
INCREMENT [BY] <i>positive-or-negative</i>	<p>Specifies how much to increment (or decrement) the current sequence value. A positive value creates an ascending sequence; a negative value makes a descending sequence. The default value is 1.</p>
MINVALUE <i>minvalue</i> NO MINVALUE	<p>Determines the minimum value a sequence can generate. If you do not specify this clause, or you specify NO MINVALUE, default values are used. The defaults are 1 and $-2^{63}-1$ for ascending and descending sequences, respectively.</p>
MAXVALUE <i>maxvalue</i> NO MAXVALUE	<p>Determines the maximum value for the sequence. If this clause is not supplied or you specify NO MAXVALUE, default values are used. The defaults are $2^{63}-1$ and -1 for ascending and descending sequences, respectively.</p>

<p>START [WITH] <i>startvalue</i></p>	<p>Specifies a specific start value of the sequence (<i>startvalue</i>). The default values are <i>minvalue</i> for ascending sequences and <i>maxvalue</i> for descending sequences.</p>
<p>CACHE <i>value</i> NO CACHE</p>	<p>Specifies how many sequence numbers are pre-allocated and stored in memory for faster access. The default is 250,000 with a minimum value of 1. The default cache size provides good efficiency for large insert or copy operations</p> <p>Specifying a cache value of 1 indicates that only one value can be generated at a time, since no cache is assigned. Alternatively, you can specify NO CACHE.</p> <p>Notes:</p> <ul style="list-style-type: none"> • If you use the CACHE clause to create a sequence, each session has its own cache on each Vertica node. • Sequences that specify a cache size that is insufficient for the number of sequence values can cause performance degradation.
<p>CYCLE NO CYCLE</p>	<p>Specifies whether the sequence can wrap when its minimum or maximum values are reached:</p> <ul style="list-style-type: none"> • CYCLE: Allows the sequence to wrap around when the maxvalue or minvalue is reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated is the minvalue or maxvalue, respectively. • NO CYCLE (default): Calls to NEXTVAL return an error after the sequence reaches its maximum/minimum value.

Privileges

To create a sequence, the user must have CREATE privilege on the schema to contain the sequence. Only the owner and superusers can initially access the sequence. All other users must be granted access to the sequence by a superuser or the owner.

To create a table with a sequence, the user must have `SELECT` privilege on the sequence and `USAGE` privilege on the schema that contains the sequence.

Note: Referencing a named sequence in a `CREATE TABLE` statement requires `SELECT` privilege on the sequence object and `USAGE` privilege on the schema of the named sequence.

Incrementing and Obtaining Sequence Values

After creating a sequence, use the `NEXTVAL` function to create a cache in which the sequence value is stored. Use the `CURRVAL` function to get the current sequence value.

You cannot use `NEXTVAL` or `CURRVAL` to act on a sequence in a `SELECT` statement:

- in a `WHERE` clause
- in a `GROUP BY` or `ORDER BY` clause
- in a `DISTINCT` clause
- along with a `UNION`
- in a subquery

Additionally, you cannot use `NEXTVAL` or `CURRVAL` to act on a sequence in:

- a subquery of `UPDATE` or `DELETE`
- a view

You can use subqueries to work around some of these restrictions. For example, to use sequences with a `DISTINCT` clause:

```
=> SELECT t.col1, shift_allocation_seq.nextval FROM (  
    SELECT DISTINCT col1 FROM av_temp1) t;
```

Removing a Sequence

Use the `DROP SEQUENCE` function to remove a sequence. You cannot drop a sequence upon which other objects depend. Sequences used in a default expression of a column cannot be dropped until all references to the sequence are removed from the default expression.

`DROP SEQUENCE ... CASCADE` is not supported.

Examples

See [Using Named Sequences](#) in the Administrator's Guide.

See Also

- [ALTER SEQUENCE](#)
- [CURRVAL](#)
- [DROP SEQUENCE](#)
- [GRANT \(Sequence\)](#)
- [NEXTVAL](#)
- [Sequence Privileges](#)

CREATE SUBNET

Identifies the subnet to which the nodes of a Vertica database belong. Use this statement when you want to configure import/export from a database to other Vertica clusters.

Syntax

```
CREATE SUBNET subnet-name WITH 'subnet prefix'
```

Parameters

<i>subnet-name</i>	A name of your choosing to assign to the subnet, where <i>subnet-name</i> conforms to conventions described in Identifiers .
<i>subnet prefix</i>	The routing prefix expressed in quad-dotted decimal representation. Refer to <code>v_monitor.network_interfaces</code> system table to get the prefix for all available IP networks.

You can then configure the database to use the subnet for import/export. (See [Identify the Database or Nodes Used for Import/Export](#) for more information.)

Privileges

You must be a dbadmin user to create a subnet.

Examples

This example shows how to create a subnet.

```
=> CREATE SUBNET mySubnet WITH '123.4.5.6';
```

CREATE TABLE

Creates a table in the logical schema. For information about creating temporary tables and external tables, see [CREATE TEMPORARY TABLE](#) and [CREATE EXTERNAL TABLE AS COPY](#), respectively.

Syntax

Create with column definitions

```
CREATE TABLE [ IF NOT EXISTS ] [schema.]table-name  
... ( column-definition[,... ] )  
... [ table-constraint ]  
... [ Load-method ]  
... [ ORDER BY table-column[,... ] ]  
... [ segmentation-spec ]  
... [ KSAFE [k-num] ]  
... [ PARTITION BY partition-expression ]  
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

Create from another table

```
CREATE TABLE [ IF NOT EXISTS ] [schema.]table-name { AS-clause | LIKE-clause }
```

AS-clause

```
... [ ( column-name-list ) ]  
... [ Load-method ]  
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]  
AS [ /*+ hint[, hint] */ ] [ AT epoch ] query [ ENCODED BY column-ref-list ]
```

LIKE-clause

```
LIKE [schema.]existing-table
...[ {INCLUDING | EXCLUDING} PROJECTIONS ]
...[ Load-method ]
...[ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

Parameters

IF NOT EXISTS	<p>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.</p> <p>The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.</p> <p>For related information, see ON_ERROR_STOP.</p>
<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre> <p>If you do not specify a schema, the table is created in the default schema.</p>
<i>table-name</i>	<p>Identifies the name of the table to create, where <i>table-name</i> conforms to conventions described in Identifiers.</p>
<i>column-definition</i>	<p>Defines a table column. A table can have up to 1600 columns.</p>
<i>table-constraint</i>	<p>Adds a constraint to table metadata.</p>
<i>Load-method</i>	<p>Specifies default load behavior for all DML operations on this table, such as INSERT and COPY, one of the following:</p> <ul style="list-style-type: none"> • AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads. • DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads. • TRICKLE: Loads data only into WOS, suitable for frequent

	<p>incremental loads.</p> <p>For details, see Choosing a Load Method in the Administrator's Guide.</p>
<p>ORDER BY <i>table-column</i> [,...]</p>	<p>Specifies columns from the SELECT list on which to sort the superprojection that is automatically created for this table. The ORDER BY clause cannot include qualifiers ASC or DESC. Vertica always stores projection data in ascending sort order.</p> <p>If you omit the ORDER BY clause, Vertica uses the SELECT list order as the projection sort order.</p> <p>This option is invalid for external tables.</p>
<p><i>segmentation-spec</i></p>	<p>Invalid for external tables, specifies how to distribute data for auto-projections of this table. Supply one of the following clauses:</p> <ul style="list-style-type: none"> • hash-segmentation-clause: Specifies to segment data evenly and distribute across cluster nodes. Vertica recommends segmenting large tables. For details, see Hash Segmentation Clause. • unsegmented-clause: Specifies to create an unsegmented projection. For details, see Unsegmented Clause. <p>If this clause is omitted, Vertica generates auto-projections with default hash segmentation.</p>
<p>KSAFE [<i>k-num</i>]</p>	<p>Specifies K-safety of auto-projections created for this table, where <i>k-num</i> must be equal to or greater than system K-safety. If you omit this option, the projection uses the system K-safety level. For general information, see K-Safety in Vertica Concepts.</p> <p>Note: This option is invalid for external tables.</p>
<p>PARTITION BY partition-expression</p>	<p>Logically divides table data storage, where partition-expression resolves to a value derived from one or more table columns. For details, see Partition Clause.</p> <p>This option is invalid for external tables.</p>

<p><i>column-name-list</i></p>	<p>Valid only when creating a table from a query (AS <i>query</i>), defines column names that map to the query output. If you omit this list, Vertica uses the query output column names. The names in <i>column-name-list</i> and queried columns must be the same in number.</p> <p>For example:</p> <pre>CREATE TABLE customer_occupations (name, profession) AS SELECT customer_name, occupation FROM customer_dimension;</pre> <p>This clause and the ENCODED BY clause are mutually exclusive. Column name lists are invalid for external tables</p>
<pre>{ INCLUDE EXCLUDE} [SCHEMA] PRIVILEGES</pre>	<p>Specifies default inheritance of schema privileges for this table:</p> <ul style="list-style-type: none"> • EXCLUDE [SCHEMA] PRIVILEGES (default) disables inheritance of privileges from the schema • INCLUDE [SCHEMA] PRIVILEGES grants the table the same privileges granted to its schema <p>For more information see Grant Inherited Privileges.</p>
<p>AS <i>query</i></p>	<p>Creates and loads a table from the results of a query, specified as follows:</p> <pre>AS [/*+hint[, hint]*/] [AT epoch] query</pre> <p>You can qualify the AS clause with one or both of the following hints:</p> <ul style="list-style-type: none"> • A load method hint: AUTO, DIRECT, or TRICKLE <div style="background-color: #f0f0f0; padding: 5px; margin: 10px 0;"> <p>Note: The CREATE TABLE statement can also specify a load method. However, this load method applies to load operations only after the table is created.</p> </div> <ul style="list-style-type: none"> • LABEL <p>For details, see Creating a Table from a Query in the Administrator's Guide.</p>
<p>ENCODED BY</p>	<p>A list of columns from the source table, where each column is</p>

<i>column-ref-list</i>	<p>qualified by one or both of the following encoding options:</p> <ul style="list-style-type: none">• ACCESSRANK <i>integer</i>: Overrides the default access rank for a column, useful for prioritizing access to a column. See Prioritizing Column Access Speed in the Administrator's Guide.• ENCODING <i>encoding-type</i>: Specifies the type of encoding to use on the column. The default encoding type is AUTO. <p>This option and <i>column-name-list</i> are mutually exclusive. This option is invalid for external tables</p>
LIKE <i>existing-table</i>	<p>Creates the table by replicating an existing table. You can qualify the LIKE clause with one of the following options:</p> <ul style="list-style-type: none">• {INCLUDING EXCLUDING} PROJECTIONS: Specifies whether to copy projections from the source table:<ul style="list-style-type: none">■ EXCLUDING PROJECTIONS (default): Do not copy projections from the source table.■ INCLUDING PROJECTIONS: Copy current projections from the source table for the new table. INCLUDING PROJECTIONS also copies all table constraints except foreign key constraints. If the table is associated with a storage policy, the policy association is also replicated.• Load-method: See description above.• {INCLUDE EXCLUDE} [SCHEMA] PRIVILEGES: See description above. <p>For details, see Replicating a Table in the Administrator's Guide.</p>

Privileges

The following privileges are required:

- CREATE privileges on the table schema
- If creating a table that includes a named sequence:
 - SELECT privilege on sequence object
 - USAGE privilege on sequence schema
- If creating a table with the LIKE clause, owner privileges on the source table

Examples

See [Creating Tables](#) in the Administrator's Guide.

See Also

- [Physical Schema](#)
- [COPY](#)
- [CREATE EXTERNAL TABLE AS COPY](#)
- [CREATE FLEX TABLE](#)
- [CREATE TEMPORARY TABLE](#)

Column-Definition

Specifies the name, data type, and constraints to be applied to a column.

Syntax

```
column-name data-type  
... [ column-constraint ]  
... [ ENCODING encoding-type ]  
... [ ACCESSRANK integer ]
```

Parameters

<i>column-name</i>	The name of a column to be created or added.
<i>data-type</i>	<p>One of the following data types:</p> <ul style="list-style-type: none">• BINARY• BOOLEAN• CHARACTER• DATE/TIME• NUMERIC <p>For information on data types, see SQL Data Types.</p> <p>Tip: When specifying the maximum column width in a CREATE TABLE statement, use the width in bytes (octets) for any of the string types. Each UTF-8 character might require four bytes, but European languages generally require a little over one byte per character, while Oriental languages generally require a little under three bytes per character.</p>
<i>column-constraint</i>	Specifies a column constraint for this column.
ENCODING <i>encoding-type</i>	Specifies the column encoding type , by default set to AUTO.
ACCESSRANK <i>integer</i>	Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking .

Example

The following example creates a table named `Employee_Dimension` and its associated superprojection in the `Public` schema. The `Employee_key` column is designated as a primary key, and RLE encoding is specified for the `Employee_gender` column definition:

```
=> CREATE TABLE Public.Employee_Dimension (  
  Employee_key          integer PRIMARY KEY NOT NULL,  
  Employee_gender       varchar(8) ENCODING RLE,  
  Courtesy_title        varchar(8),  
  Employee_first_name   varchar(64),  
  Employee_middle_initial varchar(8),  
  Employee_last_name    varchar(64)  
);
```

Column-Name-List

Used to rename columns when creating a table or temporary table from a query (CREATE TABLE AS SELECT); also used to specify the column's [encoding type](#) and access rank .

Syntax

```
column-name-list  
... [ ENCODING encoding-type ]  
... [ ACCESSRANK integer ]  
... [ GROUPED ( column-reference[,...] ) ]
```

Parameters

<i>column-name</i>	Specifies the new name for the column.
ENCODING <i>encoding-type</i>	Specifies the type of encoding to use on the column. The default encoding type is AUTO.
ACCESSRANK <i>integer</i>	Overrides the default access rank for a column, useful for prioritizing access to a column. See Prioritizing Column Access Speed in the Administrator's Guide.
GROUPED	Groups two or more columns . For detailed information, see GROUPED Clause .

Neither the data type nor column constraint can be specified for a column in the column-name-list. These are derived by the columns in the query table identified in the FROM clause. If the query output has expressions other than simple columns (for example, constants or functions) then either an alias must be specified for that expression, or all columns must be listed in the column name list.

You can supply the encoding type and access rank in either the *column-name-list* or the *column list* in the query, but not both.

Examples

The following statements are both allowed (also allowed for CREATE TEMP TABLE):

```
=> CREATE TABLE promo (state ENCODING RLE ACCESSRANK 1, zip ENCODING RLE, ...)
  AS SELECT * FROM customer_dimension
  ORDER BY customer_state, ... ;
=> CREATE TABLE promo
  AS SELECT * FROM customer_dimension
  ORDER BY customer_state
  ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING RLE ...;
```

The following statement is not allowed because encoding is specified in both *column-name-list* and ENCODED BY clause:

```
=> CREATE TABLE promo (state ENCODING RLE ACCESSRANK 1, zip ENCODING RLE, ...)
  AS SELECT * FROM customer_dimension
  ORDER BY customer_state
  ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING RLE ...;
```

Examples

The following examples show CREATE TABLE. The explanations are also true for CREATE TEMP TABLE.

The following example creates a table named `employee_dimension` and its associated superprojection in the public schema. Note that encoding-type RLE is specified for the `employee_gender` column definition:

```
=> CREATE TABLE public.employee_dimension (
  employee_key          INTEGER PRIMARY KEY NOT NULL,
  employee_gender       VARCHAR(8) ENCODING RLE,
  courtesy_title        VARCHAR(8),
  employee_first_name   VARCHAR(64),
  employee_middle_initial VARCHAR(8),
  employee_last_name    VARCHAR(64)
);
```

Using the [VMart schema](#), the following example creates a table named `promo` from a query that selects data from columns in the `customer_dimension` table. RLE encoding is specified for the state column in the column name list.

```
=> CREATE TABLE promo (
    name,
    address,
    city,
    state ENCODING RLE, income )
AS SELECT customer_name,
    customer_address,
    customer_city,
    customer_state,
    annual_income
FROM customer_dimension
WHERE annual_income > 1000000
ORDER BY customer_state, annual_income;
```

Column-Constraint

Adds a constraint to a column's metadata. See [Adding Constraints](#) in the Administrator's Guide.

Syntax

Permanent table

```
[ AUTO_INCREMENT | IDENTITY [ (args) ]
[ CONSTRAINT constraint-name ] {
...[ CHECK (expression) [ ENABLED | DISABLED ] ]
...[ [ DEFAULT default-expr ] [ SET USING using-expr ] | DEFAULT USING expr ]
...[ NULL | NOT NULL ]
...[ { PRIMARY KEY | REFERENCES table [( column )] } [ ENABLED | DISABLED ] ]
...[ UNIQUE [ ENABLED | DISABLED ] ]
}
```

Temporary table

```
[ CONSTRAINT constraint-name ] {
...| CHECK (expression) [ ENABLED | DISABLED ]
...[ DEFAULT default-expr ]
...[ NULL | NOT NULL ]
...[ { PRIMARY KEY | REFERENCES table [( column )] } [ ENABLED | DISABLED ] ]
...[ SET USING using-expr ]
...| UNIQUE [ ENABLED | DISABLED ]
}
```

Parameters

Note: A number of parameters can be qualified with the keyword ENABLED or DISABLED. For details, see [Enforcing Constraints](#) below.

AUTO_INCREMENT	Creates a table column whose values are automatically generated
----------------	---

<p>IDENTITY</p>	<p>by the database, and cannot be changed. You can set this constraint on only one table column.</p> <p>AUTO_INCREMENT is identical to IDENTITY. For details on both, see Using AUTO_INCREMENT and IDENTITY Sequences in the Administrator's Guide.</p> <p>For a general discussion of sequences, see Working with Sequence Types.</p>
<p>CONSTRAINT <i>constraint-name</i></p>	<p>Assigns a name to the constraint. Vertica recommends that you name all constraints.</p>
<p>CHECK (<i>expression</i>)</p>	<p>Adds check condition <i>expression</i>, which returns a Boolean value.</p>
<p>DEFAULT</p>	<p>Specifies this column's default value:</p> <pre>DEFAULT <i>default-expr</i></pre> <p>Vertica evaluates the DEFAULT expression and sets the column on load operations, if the operation omits a value for the column. For details about valid expressions, see Defining Column Values.</p>
<p>SET USING</p>	<p>Specifies to set values in this column from the specified expression:</p> <pre>SET USING <i>using-expr</i></pre> <p>Vertica evaluates the SET USING expression and refreshes column values only when the function REFRESH_COLUMNS is invoked. For details about valid expressions, see Defining Column Values.</p>
<p>DEFAULT USING</p>	<p>Defines the column with DEFAULT and SET USING constraints, specifying the same expression for both. DEFAULT USING columns support the same expressions as SET USING columns, and are subject to the same restrictions.</p>
<p>NULL NOT NULL</p>	<p>Specifies whether the column can contain null values:</p> <ul style="list-style-type: none"> • NULL: Allows null values in the column. If you set this constraint on a primary key column, Vertica ignores it and sets it to NOT NULL. • NOT NULL: Specifies that the column must be set to a value during insert and update operations. If the column has no default value and no value is provided, INSERT or UPDATE

	<p>returns an error.</p> <p>If you omit this constraint, the default is NULL for all columns except primary key columns, which Vertica always sets to NOT NULL.</p> <p>External tables: If you specify NOT NULL and the column contains null values, queries are liable to return errors or generate unexpected behavior. Specify NOT NULL for an external table column only if you are sure that the column does not contain nulls.</p>
PRIMARY KEY	Defines the column as the table's primary key.
REFERENCES	<p>Identifies this column as a foreign key:</p> <pre>REFERENCES <i>table</i> [<i>column</i>]</pre> <p>where <i>column</i> is the primary key in <i>table</i>. If you omit <i>column</i>, Vertica references the primary key in <i>table</i>.</p>
UNIQUE	Requires column data to be unique with respect to all table rows.

Privileges

Table owner or user WITH GRANT OPTION is grantor.

- REFERENCES privilege on table to create foreign key constraints that reference this table
- USAGE privilege on schema that contains the table

Enforcing Constraints

The following constraints can be qualified with the keyword ENABLED or DISABLED:

- PRIMARY KEY
- UNIQUE
- CHECK

If you omit `ENABLED` or `DISABLED`, Vertica determines whether to enable the constraint automatically by checking the appropriate configuration parameter:

- `EnableNewPrimaryKeysByDefault`
- `EnableNewUniqueKeysByDefault`
- `EnableNewCheckConstraintsByDefault`

For details, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Partition Clause

A table definition specifies partitioning through a `PARTITION BY` clause:

`PARTITION BY expression`

where *expression* resolves to a value derived from one or more table columns.

Requirements and Restrictions

`PARTITION BY` expressions can specify leaf expressions, functions, and operators. The following requirements and restrictions apply:

- All table projections must include all columns referenced in the `PARTITION BY` expression; otherwise, Vertica cannot resolve the expression.
- A partition clause expression can reference multiple columns, but it must resolve to a single non-null value for each row.
- All leaf expressions must be constants or table columns.
- All other expressions must be functions and operators. The following restrictions apply to functions:
 - They must be immutable—that is, they return the same value regardless of time and locale and other session- or environment-specific conditions.
 - They cannot be [aggregate functions](#).
 - They cannot be [Vertica meta-functions](#).
- A partition clause expression cannot include queries.

See Also

[Using Table Partitions](#) in the Administrator's Guide.

Table-Constraint

Adds a constraint to table metadata. You can specify table constraints with [CREATE TABLE](#), or add a constraint to an existing table with [ALTER TABLE](#). For details, see [Adding Constraints](#) in the Administrator's Guide.

Note: Adding a constraint to a table that is referenced in a view does not affect the view.

Syntax

```
[ CONSTRAINT constraint-name ]
{
... PRIMARY KEY (column[,... ]) [ ENABLED | DISABLED ]
... | FOREIGN KEY (column[,... ]) REFERENCES table [ (column[,...]) ]
... | UNIQUE (column[,...]) [ ENABLED | DISABLED ]
... | CHECK (expression) [ ENABLED | DISABLED ]
}
```

Parameters

CONSTRAINT <i>constraint-name</i>	Assigns a name to the constraint. Vertica recommends that you name all constraints.
PRIMARY KEY	<p>Defines one or more NOT NULL columns as the primary key as follows:</p> <pre>PRIMARY KEY (<i>column</i>[,...]) [ENABLED DISABLED]</pre> <p>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</p> <p>If you do not name a primary key constraint, Vertica assigns the name C_PRIMARY.</p>
FOREIGN KEY	Adds a referential integrity constraint defining one or more columns as foreign keys as follows:

	<pre>FOREIGN KEY (<i>column</i>[,...]) REFERENCES <i>table</i> [(<i>column</i>[,...])]</pre> <p>If you omit <i>column</i> references, If you omit <i>column</i>, Vertica references the primary key in <i>table</i>.</p> <p>If you do not name a foreign key constraint, Vertica assigns the name C_FOREIGN.</p>
UNIQUE	<p>Specifies that the data in a column or group of columns is unique with respect to all table rows, as follows:</p> <pre>UNIQUE (<i>column</i>[,...]) [ENABLED DISABLED]</pre> <p>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</p> <p>If you do not name a unique constraint, Vertica assigns the name C_UNIQUE.</p>
CHECK	<p>Specifies a check condition as an expression that returns a Boolean value, as follows:</p> <pre>CHECK (<i>expression</i>) [ENABLED DISABLED]</pre> <p>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</p> <p>If you do not name a check constraint, Vertica assigns the name C_CHECK.</p>

Privileges

Table owner or user WITH GRANT OPTION is grantor.

- REFERENCES privilege on table to create foreign key constraints that reference this table
- USAGE privilege on schema that contains the table

Enforcing Constraints

A table can specify whether Vertica automatically enforces a primary key, unique key or check constraint with the keyword ENABLED or DISABLED. If you omit ENABLED or DISABLED,

Vertica determines whether to enable the constraint automatically by checking the appropriate configuration parameter:

- `EnableNewPrimaryKeysByDefault`
- `EnableNewUniqueKeysByDefault`
- `EnableNewCheckConstraintsByDefault`

For details, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

Examples

The following example creates a table (`t01`) with a primary key constraint.

```
CREATE TABLE t01 (id int CONSTRAINT sampleconstraint PRIMARY KEY);  
CREATE TABLE
```

This example creates the same table without the constraint, and then adds the constraint with `ALTER TABLE ADD CONSTRAINT`

```
CREATE TABLE t01 (id int);  
CREATE TABLE  
  
ALTER TABLE t01 ADD CONSTRAINT sampleconstraint PRIMARY KEY(id);  
WARNING 2623: Column "id" definition changed to NOT NULL  
ALTER TABLE
```

The following example creates a table (`addapk`) with two columns, adds a third column to the table, and then adds a primary key constraint on the third column.

```
=> CREATE TABLE addapk (col1 INT, col2 INT);  
CREATE TABLE  
  
=> ALTER TABLE addapk ADD COLUMN col3 INT;  
ALTER TABLE  
  
=> ALTER TABLE addapk ADD CONSTRAINT col3constraint PRIMARY KEY (col3) ENABLED;  
WARNING 2623: Column "col3" definition changed to NOT NULL  
ALTER TABLE
```

Using the sample table `addapk`, check that the primary key constraint is enabled (`is_enabled is t`).

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');
```

constraint_name	column_name	constraint_type	is_enabled
col3constraint	col3	p	t

```
(1 row)
```

This example disables the constraint using `ALTER TABLE ALTER CONSTRAINT`.

```
=> ALTER TABLE addapk ALTER CONSTRAINT col3constraint DISABLED;
```

Check that the primary key is now disabled (`is_enabled` is `f`).

```
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_
name IN ('addapk');
```

```
constraint_name | column_name | constraint_type | is_enabled
-----+-----+-----+-----
col3constraint  | col3        | p               | f
(1 row)
```

For a general discussion of constraints, see [About Constraints](#). For additional examples of creating and naming constraints, see [Naming Constraints](#).

CREATE TEMPORARY TABLE

Creates a table whose data persists only during the current session. Temporary table data is not visible to other sessions.

Syntax

Create with column definitions

```
CREATE [ scope ] TEMP[ORARY] TABLE [ IF NOT EXISTS ] [schema.]table-name
... ( column-definition[,... ] )
... [ table-constraint ]
... [ ON COMMIT { DELETE | PRESERVE } ROWS ]
... [ Load-method ]
... [ NO PROJECTION ]
... [ ORDER BY table-column[,... ] ]
... [ segmentation-spec ]
... [ KSAFE [k-num] ]
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

Create from another table

```
CREATE TEMP[ORARY] TABLE [ IF NOT EXISTS ] [schema.]table-name
... [ ( column-name-list ) ]
... [ ON COMMIT { DELETE | PRESERVE } ROWS ]
... [ Load-method ]
AS [ /*+ hint[, hint] */ ] [ AT epoch ] query [ ENCODED BY column-ref-list ]
```

Parameters

<i>scope</i>	<p>Specifies visibility of the table definition:</p> <ul style="list-style-type: none"> • GLOBAL (default): The table definition is visible to all sessions, and persists until you explicitly drop the table. • LOCAL: the table definition is visible only to the session in which it is created, and is dropped when the session ends. <p>Regardless of this setting, retention of temporary table data is set by the keywords ON COMMIT DELETE and ON COMMIT PRESERVE (see below).</p> <p>For more information, see Creating Temporary Tables in the Administrator's Guide.</p>
IF NOT EXISTS	<p>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.</p> <p>The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.</p> <p>For related information, see ON_ERROR_STOP.</p>
<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre data-bbox="565 1423 1403 1486">myschema.thisDBObject</pre> <p>If you do not specify a schema, the table is created in the default schema.</p>
<i>table-name</i>	<p>Identifies the name of the table to create, where <i>table-name</i> conforms to conventions described in Identifiers.</p>
<i>column-definition</i>	<p>Defines a table column. A table can have up to 1600 columns.</p>
<i>table-constraint</i>	<p>Adds a constraint to table metadata.</p>
ON COMMIT	<p>Specifies whether data is transaction- or session-scoped:</p>

	<p style="text-align: center;"><code>ON COMMIT {PRESERVE DELETE} ROWS</code></p> <ul style="list-style-type: none"> • DELETE (default) marks the temporary table for transaction-scoped data. Vertica removes all table data after each commit. • PRESERVE marks the temporary table for session-scoped data, which is preserved beyond the lifetime of a single transaction. Vertica removes all table data when the session ends.
<p><i>Load-method</i></p>	<p>Specifies default load behavior for all DML operations on this table, such as INSERT and COPY, one of the following:</p> <ul style="list-style-type: none"> • AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads. • DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads. • TRICKLE: Loads data only into WOS, suitable for frequent incremental loads. <p>For details, see Choosing a Load Method in the Administrator's Guide.</p>
<p>NO PROJECTION</p>	<p>Prevents Vertica from creating auto-projections for this table. A superprojection is created only when data is explicitly loaded into this table.</p> <p>NO PROJECTION is invalid with the following clauses:</p> <ul style="list-style-type: none"> • ORDER BY • KSAFE • Any segmentation clause (<i>hash-segmentation-clause</i> or <i>unsegmented-clause</i>).
<p>{INCLUDE EXCLUDE} [SCHEMA] PRIVILEGES</p>	<p>Specifies default inheritance of schema privileges for this table:</p> <ul style="list-style-type: none"> • EXCLUDE [SCHEMA] PRIVILEGES (default) disables inheritance of privileges from the schema • INCLUDE [SCHEMA] PRIVILEGES grants the table the

	<p>same privileges granted to its schema</p> <p>For more information see Grant Inherited Privileges.</p>
<p>ORDER BY <i>table-column</i>[, ...]</p>	<p>Specifies columns from the SELECT list on which to sort the superprojection that is automatically created for this table. The ORDER BY clause cannot include qualifiers ASC or DESC. Vertica always stores projection data in ascending sort order.</p> <p>If you omit the ORDER BY clause, Vertica uses the SELECT list order as the projection sort order.</p> <p>This option is invalid for external tables.</p>
<p><i>segmentation-spec</i></p>	<p>Invalid for external tables, specifies how to distribute data for auto-projections of this table. Supply one of the following clauses:</p> <ul style="list-style-type: none"> • <i>hash-segmentation-clause</i>: Specifies to segment data evenly and distribute across cluster nodes. Vertica recommends segmenting large tables. For details, see Hash Segmentation Clause. • <i>unsegmented-clause</i>: Specifies to create an unsegmented projection. For details, see Unsegmented Clause. <p>If this clause is omitted, Vertica generates auto-projections with default hash segmentation.</p>
<p>KSAFE [<i>k-num</i>]</p>	<p>Specifies K-safety of auto-projections created for this table, where <i>k-num</i> must be equal to or greater than system K-safety. If you omit this option, the projection uses the system K-safety level. For general information, see K-Safety in Vertica Concepts.</p> <p>Note: This option is invalid for external tables.</p>
<p><i>column-name-list</i></p>	<p>Valid only when creating a table from a query (AS <i>query</i>), defines column names that map to the query output. If you omit this list, Vertica uses the query output column names. The names in <i>column-name-list</i> and queried columns must be the same in number.</p> <p>For example:</p> <pre>CREATE TABLE customer_occupations (name, profession)</pre>

	<pre>AS SELECT customer_name, occupation FROM customer_dimension;</pre> <p>This clause and the ENCODED BY clause are mutually exclusive. Column name lists are invalid for external tables</p>
<p><i>AS query</i></p>	<p>Creates and loads a table from the results of a query, specified as follows:</p> <pre>AS [/*+hint[, hint]*/] [AT epoch] query</pre> <p>You can qualify the AS clause with one or both of the following hints:</p> <ul style="list-style-type: none"> • A load method hint: AUTO, DIRECT, or TRICKLE <p>Note: The CREATE TABLE statement can also specify a load method. However, this load method applies to load operations only after the table is created.</p> <ul style="list-style-type: none"> • LABEL <p>For details, see Creating a Table from a Query in the Administrator's Guide.</p>
<p>ENCODED BY <i>column-ref-list</i></p>	<p>A list of columns from the source table, where each column is qualified by one or both of the following encoding options:</p> <ul style="list-style-type: none"> • ACCESSRANK <i>integer</i>: Overrides the default access rank for a column, useful for prioritizing access to a column. See Prioritizing Column Access Speed in the Administrator's Guide. • ENCODING <i>encoding-type</i>: Specifies the type of encoding to use on the column. The default encoding type is AUTO. <p>This option and <i>column-name-list</i> are mutually exclusive. This option is invalid for external tables</p>

Privileges

The following privileges are required:

- CREATE privileges on the table schema
- If creating a temporary table that includes a named sequence:
 - SELECT privilege on sequence object
 - USAGE privilege on sequence schema

Restrictions

- Queries on temporary tables are subject to the same restrictions on SQL support as persistent tables.
- You cannot add projections to non-empty, global temporary tables (ON COMMIT PRESERVE ROWS). Make sure that projections exist before you load data. See [Auto-Projections](#) in the Administrator's Guide.
- While you can add projections for temporary tables that are defined with ON COMMIT DELETE ROWS specified, be aware that you might lose all data.
- Moveout and mergeout operations cannot be used on session-scoped temporary data.
- In general, session-scoped temporary table data is not visible using system (virtual) tables.
- Temporary tables do not recover. If a node fails, queries that use the temporary table also fail. Restart the session and populate the temporary table.

See Also

- [ALTER TABLE](#)
- [CREATE TABLE](#)
- [Creating Temporary Tables](#)

CREATE TEXT INDEX

Creates a text index used to perform text searches.

Syntax

```
CREATE TEXT INDEX [schema.]txtindex_name
.. ON [schema.]source-table (unique-id, text-field [, column-name, ...])
.. [STEMMER {stemmer-name(stemmer-input-data-type)| NONE}]
.. [TOKENIZER tokenizer-name(tokenizer-input-data-type)];
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre> <p>If you do not specify a schema, the table is created in the default schema.</p>
<i>txtindex-name</i>	The text index name.
<i>source-table</i>	The source table to index.
<i>unique-id</i>	The name of the column in the source table that contains a unique identifier. Any data type is permissible. The column must be the primary key in the source table.
<i>text-field</i>	<p>The name of the column in the source table that contains the text field. Valid data types are:</p> <ul style="list-style-type: none"> • CHAR • VARCHAR • LONG VARCHAR • VARBINARY • LONG VARBINARY <p>Nulls are allowed.</p>
<i>column-name</i>	The name of a column or columns to be included as additional columns.

<i>stemmer-name</i>	The name of the stemmer.
<i>stemmer-input-data-type</i>	The input data type of the <i>stemmer-name</i> function.
<i>tokenizer-name</i>	Specifies the name of the tokenizer.
<i>tokenizer-input-data-type</i>	This value is the input data type of the <i>tokenizer-name</i> function. It can accept any number of arguments. If a Vertica Tokenizers is used, then this parameter can be omitted.

Privileges

The index automatically inherits the query permissions of its parent table. The table owner and dbadmin will be allowed to create and/or modify the indices.

Important: Do not alter the contents or definitions of the text index. If the contents or definitions of the text index are altered, then the results will not appropriately match the source table.

Requirements

- Requires there be a column with a unique identifier set as the primary key.
- The source table must have an associated projection, and must be both sorted and segmented by the primary key.

Behavior

If data within a table is partitioned, then an extra column appears in the text index, showing the partition.

Examples

The following example shows how to create a text index with an additional unindexed column on the table `t_log` using the `CREATE TEXT INDEX` statement:

```
=> CREATE TEXT INDEX t_log_index ON t_log (id, text, day_of_week);
CREATE INDEX

=> SELECT * FROM t_log_index;
      token          | doc_id | day_of_week
-----+-----+-----
'catalog             |      1 | Monday
'dbadmin'            |      2 | Monday
2014-06-04           |      1 | Monday
2014-06-04           |      2 | Monday
2014-06-04           |      3 | Monday
2014-06-04           |      4 | Monday
2014-06-04           |      5 | Monday
2014-06-04           |      6 | Monday
2014-06-04           |      7 | Monday
2014-06-04           |      8 | Monday
45035996273704966   |      3 | Tuesday
45035996273704968   |      4 | Tuesday
<INFO>               |      1 | Tuesday
<INFO>               |      6 | Tuesday
<INFO>               |      7 | Tuesday
<INFO>               |      8 | Tuesday
<WARNING>            |      2 | Tuesday
<WARNING>            |      3 | Tuesday
<WARNING>            |      4 | Tuesday
<WARNING>            |      5 | Tuesday

...

(97 rows)
```

The following example shows a text index, `tpart_index`, created from a partitioned source table:

```
=> SELECT * FROM tpart_index;
      token          | doc_id | partition
-----+-----+-----
0                  |      4 | 2014
0                  |      5 | 2014
11:00:49.568      |      4 | 2014
11:00:49.568      |      5 | 2014
11:00:49.569      |      6 | 2014
<INFO>            |      6 | 2014
<WARNING>         |      4 | 2014
<WARNING>         |      5 | 2014
Database          |      6 | 2014
Execute:          |      6 | 2014
Object            |      4 | 2014
Object            |      5 | 2014
[Catalog]         |      4 | 2014
[Catalog]         |      5 | 2014
'catalog'         |      1 | 2013
'dbadmin'         |      2 | 2013
0                  |      3 | 2013
11:00:49.568      |      1 | 2013
11:00:49.568      |      2 | 2013
11:00:49.568      |      3 | 2013
11:00:49.570      |      7 | 2013
```

```
11:00:49.571      |      8 |      2013  
45035996273704966 |      3 |      2013
```

...

(89 rows)

See Also

- [Using Text Search](#)
- [DROP TEXT INDEX](#)

CREATE USER

Adds a name to the list of authorized database users.

Note: New users lack default access to schema PUBLIC. Be sure to assign new users USAGE privileges to the PUBLIC schema ([GRANT USAGE ON SCHEMA PUBLIC](#))

Syntax

```
CREATE USER name [ user-parameter[,...] ]
```

user-parameter

```
ACCOUNT { LOCK | UNLOCK }  
GRACEPERIOD limit  
IDENTIFIED BY 'password'  
IDLESESSIONTIMEOUT limit  
MAXCONNECTIONS limit  
MEMORYCAP limit  
PASSWORD EXPIRE  
PROFILE profile  
RESOURCE POOL pool-name  
RUNTIMECAP expression  
SEARCH_PATH path  
TEMPSPACECAP limit
```

Parameters

<p><i>name</i></p>	<p>Specifies the name of the new user. Names that contain special characters must be double-quoted. To enforce case-sensitivity, use double-quotes.</p> <p>For details on name requirements, see Creating a Database Name and Password.</p>
<p>ACCOUNT { LOCK UNLOCK }</p>	<p>Locks or unlocks a user's access to the database:</p> <ul style="list-style-type: none"> • UNLOCK (default) • LOCK prevents a new user from logging in. This can be useful when creating an account for a user who does not need immediate access. <p>Tip: To automate account locking, set a maximum number of failed login attempts with CREATE PROFILE.</p>
<p>GRACEPERIOD <i>Limit</i></p>	<p>Specifies how long a user query can block on any session socket, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> • NONE (default): Removes any grace period previously set on session queries. • '<i>interval</i>': Specifies as an interval the maximum grace period for current session queries, up to 20 days. <p>For details, see Handling Session Socket Blocking.</p>
<p>IDENTIFIED BY '<i>password</i>'</p>	<p>Sets the new user's password, where <i>password</i> must conform to the password complexity policy set by the user's profile.</p> <p>If you supply an empty string or omit this clause, the user is assigned no password and is not prompted for one when connecting.</p> <p>For details, see Password Guidelines and Creating a Database Name and Password.</p>
<p>IDLESESSIONTIMEOUT <i>Limit</i></p>	<p>The length of time the system waits before disconnecting an idle session, where <i>Limit</i> is one of the following:</p>

	<ul style="list-style-type: none"> • NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user. • '<i>interval</i>' An interval value, up to one year. <p>For details, see Managing Client Connections.</p>
<p>MAXCONNECTIONS <i>Limit</i></p>	<p>Indicates the maximum number of connections the user can have to the server, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> • NONE (default): No limit set. If you omit this parameter, the user can have an unlimited number of connections. • '<i>integer</i>' ON NODE: Sets the maximum number of connections to each node to <i>integer</i>. • '<i>integer</i>' ON DATABASE: Sets the maximum number of connections across the database cluster to <i>integer</i>. <p>For details, see Managing Client Connections.</p>
<p>MEMORYCAP <i>Limit</i></p>	<p>Specifies how much memory can be allocated to user requests, where <i>Limit</i> is specified in this format:</p> <ul style="list-style-type: none"> • NONE (default): No limit • '<i>max-expression</i>': A string value that specifies the memory limit, one of the following: <ul style="list-style-type: none"> ■ <i>int%</i> — Expresses the maximum as a percentage of total memory available to the Resource Manager, where <i>int</i> is an integer value between 0 and 100. For example: MEMORYCAP '40%' ■ <i>int{K M G T}</i> — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example: MEMORYCAP '10G'
<p>PASSWORD EXPIRE</p>	<p>Forces immediate expiration of the user's password. The user must change the password on the next login.</p> <p>Note: PASSWORD EXPIRE has no effect when using</p>

	external password authentication methods such as LDAP or Kerberos.
PROFILE <i>profile</i>	<p>Assigns the user to a profile, where <i>profile</i> is one of the following:</p> <ul style="list-style-type: none"> • DEFAULT (default): Assigns the default profile to this user. If you omit this parameter, the user is assigned to the default profile. • <i>profile-name</i>: The profile assigned to this user, which includes the user's password policy.
RESOURCE POOL <i>pool-name</i>	Assigns a default resource pool to this user. The user must also be granted privileges to this pool , unless privileges to the pool are set to PUBLIC.
RUNTIMECAP <i>Limit</i>	<p>Specifies how long this user's queries can execute, where <i>Limit</i> is one of the following:</p> <ul style="list-style-type: none"> • NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user. • '<i>interval</i>' An interval value, up to one year. <p>A query's runtime limit can be set at three levels: the user's runtime limit, the user's resource pool, and the session setting. For more information, see Setting a Runtime Limit for Queries in the Administrator's Guide.</p>
SEARCH_PATH <i>path</i>	<p>Specifies the user's default search path that tells Vertica which schemas to search for unqualified references to tables and UDFs, where <i>path</i> is one of the following:</p> <ul style="list-style-type: none"> • DEFAULT (default): Sets the search path as follows: "\$user", public, v_catalog, v_monitor, v_internal • A comma-delimited list of schemas. <p>For details, see Setting Search Paths in the Administrator's Guide.</p>
TEMPSPACECAP <i>Limit</i>	Limits how much temporary file storage is available for user requests, where <i>Limit</i> is one of the following:

	<ul style="list-style-type: none">• NONE (default): No limit• '<i>max-expression</i>': A string value that specifies the storage limit, one of the following:<ul style="list-style-type: none">■ <i>int</i>% — Expresses storage as a percentage of total file space is available, where <i>int</i> is an integer value between 0 and 100. For example: TEMPSPACECAP '40%'■ <i>int</i>{K M G T} — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example: TEMPSPACECAP '10G'
--	---

Privileges

Superusers

User Name Best Practices

Vertica database user names are logically separate from user names of the operating system in which the server runs. If all the users of a particular server also have accounts on the server's machine, it makes sense to assign database user names that match their operating system user names. However, a server that accepts remote connections might have many database users with no local operating system account. In this case, there is no need to connect database and system user names.

Examples

```
=> CREATE USER Fred;  
=> GRANT USAGE ON SCHEMA PUBLIC to Fred;
```

See Also

- [ALTER USER](#)
- [DROP USER](#)

CREATE VIEW

Defines a view. Views are read only, so they do not support insert, update, delete, or copy operations.

Syntax

```
CREATE [ OR REPLACE ] VIEW view-name [ (column-name[,...]) ]  
[ {INCLUDE|EXCLUDE} [SCHEMA] PRIVILEGES ] AS query
```

Parameters

OR REPLACE	Specifies to overwrite the existing view <i>view-name</i> . If you omit this option and <i>view-name</i> already exists, CREATE VIEW returns an error.
<i>view-name</i>	Specifies the name of the view to create, where <i>view-name</i> conforms to conventions described in Identifiers . <i>view-name</i> must not be the name of an existing table, view, or projection.
<i>column-name</i> [,...]	A list of names to use as view column names. Vertica maps view column names to query columns according to the order of their respective lists. By default, the view uses column names as they are specified in the query. Each view can contain up to 1600 columns.
<i>query</i>	A SELECT statement that the temporary view executes. The SELECT statement can reference tables, temporary tables, and other views.
{INCLUDE EXCLUDE}	Specifies whether this view inherits schema privileges:

[SCHEMA] PRIVILEGES

- **INCLUDE PRIVILEGES:** Inherit schema privileges.
- **EXCLUDE PRIVILEGES:** Do not inherit schema privileges.

For details, see [Grant Inherited Privileges](#).

Privileges

See [Creating Views](#)

Examples

The following example shows how to create a view that contains data from multiple tables.

```
=> CREATE VIEW temp_t0 AS SELECT * from t0_p1 UNION ALL
    SELECT * from t0_p2 UNION ALL
    SELECT * from t0_p3 UNION ALL
    SELECT * from t0_p4 UNION ALL
    SELECT * from t0_p5;
```

See Also

- [ALTER VIEW](#)
- [CREATE LOCAL TEMPORARY VIEW](#)
- [Creating Views](#)
- [DROP VIEW](#)
- [GRANT \(View\)](#)
- [REVOKE \(View\)](#)

DELETE

Removes the specified rows from a table and returns a count of the deleted rows. A count of 0 is not an error, but indicates that no rows matched the condition. An unqualified DELETE

statement (omits a WHERE clause) removes all rows but leaves intact table columns, projections, and constraints.

DELETE supports subqueries and joins, so you can delete values in a table based on values in other tables.

Syntax

```
DELETE [ /*+ hint[, hint] */ ] FROM [schema.]table-name [ where-clause ]
```

Parameters

<code>/*+ hint [, hint] */</code>	One or both of the following hints: <ul style="list-style-type: none">• DIRECT• LABEL
<code>schema</code>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<code>table-name</code>	Any table, including temporary tables.
<code>where-clause</code>	Specifies which rows to mark for deletion. If you omit this clause, DELETE behavior varies depending on whether the table is persistent or temporary. See below for details.

Privileges

Table owner or user with GRANT OPTION is grantor.

- DELETE privilege on table
- USAGE privilege on the schema of the target table
- SELECT privilege on a table when the DELETE statement includes a WHERE or SET clause that specifies columns from that table.

Restrictions

You cannot execute DELETE on a projection.

Deleting from Persistent Tables

Unlike [TRUNCATE TABLE](#), DELETE does not delete data from disk storage. Rather, it marks rows for deletion in the WOS. These rows are no longer valid in the current epoch. By default, DELETE uses WOS. If WOS fills up, the operation overflows to the ROS.

Deleting from a Temporary Table

DELETE execution on temporary tables varies, depending on whether the table was created with `ON COMMIT DELETE ROWS` (default) or `ON COMMIT PRESERVE ROWS`:

- If DELETE contains a WHERE clause that specifies which rows to remove, behavior is identical: DELETE marks the rows for deletion. In both cases, you cannot roll back to an earlier savepoint.
- If DELETE omits a WHERE clause and the table was created with `ON COMMIT PRESERVE ROWS`, Vertica marks all table rows for deletion. If the table was created with `ON COMMIT DELETE ROWS`, DELETE behaves like [TRUNCATE TABLE](#) and removes all rows from storage.

Note: If you issue an unqualified DELETE statement on a temporary table created with `ON COMMIT DELETE ROWS`, Vertica removes all rows from storage but does not end the transaction.

Examples

The following command removes all rows from temporary table temp1:

```
=> DELETE FROM temp1;
```

The following command deletes all records from anchor table T where $C1 = C2 - C1$.

```
=> DELETE FROM T WHERE C1=C2-C1;
```

The following command deletes all records from the customer table in the retail schema where the state attribute is in MA or NH:

```
=> DELETE FROM retail.customer WHERE state IN ('MA', 'NH');
```

For examples that show how to nest a subquery within a DELETE statement, see [Subqueries in UPDATE and DELETE](#) in Analyzing Data.

See Also

- [DROP TABLE](#)
- [TRUNCATE TABLE](#)
- [Removing Table Data](#)
- [Best Practices for DELETE and UPDATE](#)

Directed Query Statements

The following statements let you create and manage directed queries:

ACTIVATE DIRECTED QUERY	Activates a directed query.
CREATE DIRECTED QUERY	Saves an association between an input query and a query that is annotated with optimizer hints.
DEACTIVATE DIRECTED QUERY	Deactivates one or more directed queries.
DROP DIRECTED QUERY	Removes a directed query from the database.
GET DIRECTED QUERY	Returns a list of directed queries stored in the database for a given input query.
SAVE QUERY	Saves an input query to associate with a custom directed query.

ACTIVATE DIRECTED QUERY

Activates a directed query and makes it available to the query optimizer across all sessions.

Syntax

```
ACTIVATE DIRECTED QUERY query-name
```

Parameters

<i>query-name</i>	Identifies the directed query to activate. To obtain identifiers for directed queries, use GET DIRECTED QUERY , or query the system table V_CATALOG.DIRECTED_QUERIES .
-------------------	---

Privileges

Superuser

Activation Life Cycle

After you activate a directed query, it remains active until it is explicitly deactivated by [DEACTIVATE DIRECTED QUERY](#) or removed from storage by [DROP DIRECTED QUERY](#). If a directed query is active at the time of database shutdown, Vertica automatically reactivates it when you restart the database.

Examples

See [Activating and Deactivating Directed Queries](#).

CREATE DIRECTED QUERY

Saves an association between an input query and a query that is annotated with optimizer hints.

Syntax

Optimizer-generated

```
CREATE DIRECTED QUERY OPT[IMIZER] directedqueryID [COMMENT 'comments']  
input-query
```

User-generated (custom)

```
CREATE DIRECTED QUERY CUSTOM directedqueryID [COMMENT 'comments']  
annotated-query
```

Parameters

OPT[IMIZER]	Directs the query optimizer to generate an annotated query from <i>input-query</i> , and associate both in the new directed query.
CUSTOM	Specifies to associate <i>annotated-query</i> with the query previously specified by SAVE QUERY .
<i>directedqueryID</i>	A unique identifier for the directed query, a string that conforms to conventions described in Identifiers .
COMMENT ' <i>comments</i> '	<p>Comments about the directed query, up to 128 characters. Comments can be useful for future reference—for example, explain why a given directed query was created.</p> <p>If you omit this argument, Vertica inserts one of the following comments:</p> <ul style="list-style-type: none"> Optimizer-generated directed query Custom directed query
<i>input-query</i>	The input query to associate with an optimizer-generated directed query. The input query supports only one optimizer hint, IGNORECONST .
<i>annotated-query</i>	A query with embedded optimizer hints to associate with the input query most recently saved with SAVE QUERY .

Privileges

Superuser

Description

`CREATE DIRECTED QUERY` associates an input query with a query annotated with optimizer hints. It stores the association under a unique identifier. `CREATE DIRECTED QUERY` has two variants:

- `CREATE DIRECTED QUERY OPTIMIZER` directs the query optimizer to generate annotated SQL from the specified input query. The annotated query contains hints that the optimizer can use to recreate its current query plan for that input query.
- `CREATE DIRECTED QUERY CUSTOM` specifies an annotated query supplied by the user. Vertica associates the annotated query with the input query specified by the last `SAVE QUERY` statement.

In both cases, Vertica associates the annotated query and input query, and registers their association in the system table `V_CATALOG.DIRECTED_QUERIES` under `query_name`.

Caution: Vertica associates a saved query and directed query without checking whether the two are compatible. Be careful to sequence `SAVE QUERY` and `CREATE DIRECTED QUERY CUSTOM` so the saved and directed queries are correctly matched.

See Also

[Creating Directed Queries](#)

DEACTIVATE DIRECTED QUERY

Deactivates one or more directed queries previously activated by `ACTIVATE DIRECTED QUERY`.

Syntax

```
DEACTIVATE DIRECTED QUERY {query-name|input-query}
```

Arguments

<i>query-name</i>	Identifies the directed query to deactivate. To obtain identifiers for directed queries, use GET DIRECTED QUERY , or query the system table V_CATALOG.DIRECTED_QUERIES .
<i>input-query</i>	The input query of the directed queries to deactivate. Use this argument to deactivate multiple direct queries that map to the same input query.

Privileges

Superuser

DROP DIRECTED QUERY

Removes a directed query from the database. If the directed query is active, Vertica deactivates it before removal.

Syntax

```
DROP DIRECTED QUERY directedqueryID
```

Arguments

<i>directedqueryID</i>	Identifies the directed query to remove from the database. To obtain identifiers for directed queries, use GET DIRECTED QUERY , or query the system table V_CATALOG.DIRECTED_QUERIES .
------------------------	--

Privileges

Superuser

GET DIRECTED QUERY

Returns a list of directed queries that map to the specified input query

Syntax

```
GET DIRECTED QUERY input-query
```

Parameters

<i>input-query</i>	An input query that is associated with one or more directed queries.
--------------------	--

Returns

A list of one or more directed queries that map to the specified input query. Each list item includes six fields:

query-name	A unique identifier that is associated with this directed query and used by statements such as ACTIVATE DIRECTED QUERY .
is_active	A Boolean field that specifies whether the directed query is active.
vertica_version	The Vertica version that was current when this directed query was created.
comment	A user-supplied comment specified on creation of the direct query.
creation_date	The creation timestamp of this directed query.
annotated_query	The annotated query that was saved with CREATE DIRECTED QUERY .

Privileges

None required

Examples

See [Getting Directed Queries](#) in the Administrator's Guide.

See Also

[V_CATALOG.DIRECTED_QUERIES](#)

SAVE QUERY

Saves an input query to associate with a custom directed query.

Syntax

```
SAVE QUERY input-query
```

Parameters

<i>input-query</i>	The input query to associate with a custom directed query. The input query supports only one optimizer hint, IGNORECONST .
--------------------	--

Privileges

Superuser

Description

`SAVE QUERY` saves the specified input query for use by the next invocation of [CREATE DIRECTED QUERY CUSTOM](#). `CREATE DIRECTED QUERY CUSTOM` pairs the saved query with its annotated query argument to create a directed query. Both statements must be issued in the same user session.

The saved query remains available until the one of the following events occurs:

- The next invocation of `CREATE DIRECTED QUERY`, whether invoked with `CUSTOM` or `OPTIMIZER`.
- Another invocation of `SAVE QUERY`.
- The session ends.

Caution: Vertica associates a saved query with a directed query without checking whether the input and annotated queries are compatible. Be careful to sequence `SAVE QUERY` and `CREATE DIRECTED QUERY CUSTOM` so the saved and directed queries are correctly matched.

Examples

See [Custom Directed Queries](#).

DISCONNECT

Closes a previously established connection to another Vertica database. You must have previously used the `CONNECT` statement to perform a `COPY FROM VERTICA` or `EXPORT TO VERTICA` statement.

Syntax

```
DISCONNECT database-name
```

Parameters

<i>database-name</i>	The name of the database whose connection to close.
----------------------	---

Privileges

No special permissions required.

Example

```
=> DISCONNECT ExampleDB;  
DISCONNECT
```

See Also

- [CONNECT](#)
- [COPY FROM VERTICA](#)
- [EXPORT TO VERTICA](#)

DROP ACCESS POLICY

Removes an access policy from a column or row. If you attempt to run a [DROP TABLE](#) statement on a table that contains an access policy, the following message appears:

```
NOTICE 4927: The AccessPolicy depends on Table tablename  
ROLLBACK 3128: DROP failed due to dependencies  
DETAIL: Cannot drop Table tablename because other objects depend on it  
Projection projection has column column_name as part of its sort order  
Projection public.p_1 has column column_name as part of its sort order  
HINT: Use DROP .. CASCADE to drop or modify the dependent objects
```

Syntax

```
DROP ACCESS POLICY ON tablename  
FOR { COLUMN columnname | ROWS};
```

Parameters

<i>tablename</i>	The name of the table that contains the column access policy you want to remove.
<i>columnname</i>	The name of the column that contains the access policy you want to remove.

Privileges

You must be a dbadmin user to drop an access policy.

Examples

These examples show various cases where you can drop an access policy.

Drop Column Access Policy from Customer Table for a Specified Column

```
=> DROP ACCESS POLICY ON customer FOR COLUMN Customer_Number;
```

Drop Access Policy and Table Using the CASCADE Keyword with DROP TABLE

```
=> DROP TABLE <tablename> CASCADE;
```

Drop Row Access Policy on customer_info table

```
=> DROP ACCESS POLICY ON customer_info FOR ROWS;
```

DROP AGGREGATE FUNCTION

Drops a User Defined Aggregate Function (UDAF) from the Vertica catalog.

Syntax

```
DROP AGGREGATE FUNCTION [schema.]function-name [, ...]  
... ( [ [argname] argtype[, ...] ] )
```

Parameters

schema	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
--------	---

<i>function-name</i>	Specifies a name of the SQL function to drop. If the function name is schema-qualified, the function is dropped from the specified schema (as noted above).
<i>argname</i>	Specifies the name of the argument, typically a column name.
<i>argtype</i>	Specifies the data type for argument(s) that are passed to the function. Argument types must match Vertica type names. See SQL Data Types .

Notes

- To drop a function, you must specify the argument types because several functions might share the same name with different parameters.
- Vertica does not check for dependencies, so if you drop a SQL function where other objects reference it (such as views or other SQL functions), Vertica returns an error when those objects are used and not when the function is dropped.

Privileges

One of the following:

- Superuser
- Owner

Example

The following command drops the `ag_avg` function:

```
=> DROP AGGREGATE FUNCTION ag_avg(numeric);  
DROP AGGREGATE FUNCTION
```

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [CREATE AGGREGATE FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Using User-Defined SQL Functions](#)
- [Developing User-Defined Extensions \(UDxs\)](#)
- [Aggregate Functions \(UDAFs\)](#)

DROP AUTHENTICATION

Drops an authentication method that has been created but not granted to any users. If you try to drop an authentication method that has been granted to one or more users, the DROP operation fails. To successfully drop an authentication method that is currently granted to a user, use the CASCADE keyword.

Syntax

```
DROP AUTHENTICATION auth_method_name [ CASCADE ]
```

Parameters

<i>auth_method_name</i>	Name of the authentication method to drop. Type: VARCHAR (not case sensitive)
-------------------------	---

Privileges

Must have DBADMIN privileges.

Examples

To delete an authentication record for md5_auth, use the following command:

```
=> DROP AUTHENTICATION md5_auth;
```

To delete an authentication record for a method that has been granted to a user, use the CASCADE keyword:

```
=> CREATE AUTHENTICATION localpwd METHOD 'password' LOCAL;  
=> GRANT AUTHENTICATION localpwd TO jsmith;  
=> DROP AUTHENTICATION localpwd CASCADE;
```

- [ALTER AUTHENTICATION](#)
- [CREATE AUTHENTICATION](#)
- [GRANT \(Authentication\)](#)
- [REVOKE \(Authentication\)](#)

DROP FAULT GROUP

A drop operation removes the specified fault group and its child fault groups, placing all nodes under the parent of the dropped fault group.

To drop all fault groups, use [ALTER DATABASE..DROP ALL FAULT GROUP](#) syntax.

To add an orphaned node back to a fault group, you must manually re-assign it to a new or existing fault group using a combination of [CREATE FAULT GROUP](#) and [ALTER FAULT GROUP..ADD NODE](#) statements.

Tip: For a list of all fault groups defined in the cluster, query the [V_CATALOG.FAULT_GROUPS](#) system table.

Syntax

```
DROP FAULT GROUP name
```

Parameters

<i>name</i>	Specifies the name of the fault group to drop.
-------------	--

Privileges

Must be a superuser to drop a fault group.

Example

The following example drops the group2 fault group from the cluster:

```
exampledb=> DROP FAULT GROUP group2;  
DROP FAULT GROUP
```

See Also

- [V_CATALOG.FAULT_GROUPS](#) and [V_CATALOG.CLUSTER_LAYOUT](#)
- [Fault Groups](#) in the Administrator's Guide
- [High Availability With Fault Groups](#) in Vertica Concepts

DROP FUNCTION

Drops a SQL function or User Defined Function (UDF) from the Vertica catalog.

Syntax

```
DROP FUNCTION [schema.]function-name[,...] ( [ arg-List ] )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>function-name</i>	The SQL function to drop.
<i>arg-List</i>	<p>A comma-delimited list of arguments as defined for this function when it was created, specified as follows:</p> <pre>[<i>arg-name</i>] <i>arg-type</i>[,...]</pre> <p>where <i>arg-name</i> optionally qualifies <i>arg-type</i>:</p> <ul style="list-style-type: none">• <i>arg-name</i> is typically a column name.• <i>arg-type</i> is the name of an SQL data type supported by Vertica.

Privileges

One of the following:

- Superuser
- Schema or function owner

Requirements

- To drop a function, you must specify the argument types because several functions might share the same name with different parameters.
- Vertica does not check for dependencies, so if you drop a SQL function where other objects reference it (such as views or other SQL functions), Vertica returns an error when those objects are used and not when the function is dropped.

Example

The following command drops the `zerowhennull` function in the `macros` schema:

```
=> DROP FUNCTION macros.zerowhennull(x INT);  
DROP FUNCTION
```

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [CREATE FUNCTION \(SQL Functions\)](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Using User-Defined SQL Functions](#)

DROP SOURCE

Drops a User Defined Load Source function from the Vertica catalog.

Syntax

```
DROP SOURCE [schema.]source-name()
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>source-name</i> ()	<p>Specifies the source function to drop. You must append empty parentheses to the function name.</p>

Privileges

One of the following:

- Superuser
- Source function owner

Example

The following command drops the `curl` source function:

```
=> DROP SOURCE curl();  
DROP SOURCE
```

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [CREATE SOURCE](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Load \(UDLs\)](#)

DROP FILTER

Drops a User Defined Load Filter function from the Vertica catalog.

Syntax

```
DROP FILTER [schema.]filter-name()
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>filter-name</i> ()	<p>Specifies the filter function to drop. You must append empty parentheses to the function name.</p>

Privileges

Only the superuser or owner can drop the filter function.

Example

The following command drops the Iconverter filter function::

```
=> drop filter Iconverter();  
DROP FILTER
```

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [CREATE FILTER](#)
- [GRANT \(User Defined Extension\)](#)

- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Load \(UDLs\)](#)

DROP PARSER

Drops a User Defined Load Parser function from the Vertica catalog.

Syntax

```
DROP PARSER[schema.]parser-name()
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>parser-name()</i>	<p>Specifies the parser function to drop. You must append empty parentheses to the function name.</p>

Privileges

Only the superuser or owner can drop the parser function.

Example

The following command drops the BasicIntegerParser parser function:

```
=> DROP PARSER BasicIntegerParser();  
DROP PARSER
```

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [CREATE PARSER](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [USER_FUNCTIONS](#)
- [Load \(UDLs\)](#)

DROP LIBRARY

Removes a shared library from the database. The library file is deleted from managed directories on the Vertica nodes. The user defined functions (UDFs) in the library are no longer available. See [Developing User-Defined Extensions \(UDxs\)](#) in Extending Vertica for details.

Syntax

```
DROP LIBRARY [schema.]Library-name [CASCADE]
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>Library-name</i>	The name of the library to drop, the same name used in CREATE LIBRARY to load the library.
CASCADE	Drops any functions that were defined using the library. <code>DROP LIBRARY</code> fails if <code>CASCADE</code> is omitted and one or more UDFs use the target library.

Privileges

Superuser

Example

To drop the library MyFunctions:

```
=> DROP LIBRARY MyFunctions CASCADE;
```

DROP MODEL

Removes a model from the Vertica database. You can drop multiple models at one time.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
DROP MODEL [IF EXISTS] [[db-name.]schema.]model-name
```

Parameters

<code>IF EXISTS</code>	Specifies not to report an error if one or more of the models to drop does not exist. This clause is useful in SQL scripts where you want to drop a model if it exists before recreating it.
<code>[<i>db-name.</i>]<i>schema</i></code>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema</pre>
<code><i>model-name</i></code>	The model to drop.

Privileges

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

Examples

This example shows how you can remove a model.

```
=> DROP MODEL mySVModel;  
DROP MODEL
```

DROP NETWORK INTERFACE

Removes a network interface from Vertica. You can use the CASCADE option to also remove the network interface from any node definition. (See [Identify the Database or Nodes Used for Import/Export](#) for more information.)

Syntax

```
DROP NETWORK INTERFACE network-interface-name [CASCADE]
```

Parameters

The parameters are defined as follows:

<i>network-interface-name</i>	The network interface you want to remove.
-------------------------------	---

Privileges

Must be a superuser to drop a network interface.

Examples

This example shows how to drop a network interface.

```
=> DROP NETWORK INTERFACE myNetwork;
```

DROP NOTIFIER

Drops a push-based notifier created by [CREATE NOTIFIER](#).

Syntax

```
DROP NOTIFIER notifier-name
```

Parameters

<i>notifier-name</i>	This notifier's unique identifier.
----------------------	------------------------------------

DROP PROCEDURE

Removes an external procedure from Vertica. Only the reference to the procedure is removed. The external file remains in the *database/procedures* directory of each database node.

Syntax

```
DROP PROCEDURE [schema.]procedure-name( [ arg-List ] )
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database,
---------------	---

	include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>procedure-name</i>	Specifies the procedure to drop.
<i>arg-List</i>	A comma-delimited list of arguments defined for this procedure when it was created, specified as follows: <pre>[arg-name] arg-type[, ...]</pre> where <i>arg-name</i> optionally qualifies <i>arg-type</i> . If no arguments are defined for this procedure, specify empty parentheses.

Privileges

- Superuser or procedure owner
- USAGE privilege on schema, or schema owner

Example

```
=> DROP PROCEDURE helloplanet(arg1 varchar);
```

See Also

[CREATE PROCEDURE](#)

DROP PROFILE

Removes a profile from the database. Only the superuser can drop a profile.

Syntax

```
DROP PROFILE name [, ...] [ CASCADE ]
```

Parameters

<i>name</i>	The name of one or more profiles (separated by commas) to be removed.
CASCADE	Moves all users assigned to the profile or profiles being dropped to the DEFAULT profile. If you do not include CASCADE in the DROP PROFILE command and a targeted profile has users assigned to it, the command returns an error.

Privileges

Must be a superuser to drop a profile.

Notes

You cannot drop the DEFAULT profile.

Example

```
=> DROP PROFILE sample_profile;
```

See Also

- [ALTER PROFILE](#)
- [CREATE PROFILE](#)

DROP PROJECTION

Marks a projection to drop from the catalog so it is unavailable to user queries.

Syntax

```
DROP PROJECTION { [schema.]proj-name [ , ... ] } ... [ RESTRICT | CASCADE ]
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>projection-name</i>	<p>Specifies a projection to drop:</p> <ul style="list-style-type: none">• If the projection is unsegmented, all projection replicas in the database cluster are dropped.• If the projection is segmented, drop all buddy projections by specifying the projection base name. You can also specify the name of a specific buddy projection as long as dropping it so does not violate system K-safety. <p>See Projection Naming for projection name conventions.</p>
RESTRICT CASCADE	<p>Specifies whether to drop the projection when it contains objects:</p> <ul style="list-style-type: none">• RESTRICT (default): Drop the projection only if it contains no objects.• CASCADE: Drop the projection even if it contains objects.

Privileges

To drop a projection, two requirements apply:

- Ownership of the projection's anchor table
- Ownership of the projection's schema, or USAGE privileges on the schema

Restrictions

The following restrictions apply to dropping a projection:

- The projection cannot be the anchor table's superprojection.
- You cannot drop a buddy projection if doing so violates system K-safety.
- Another projection must be available to enforce the same primary or unique key constraint.

See Also

- [CREATE PROJECTION](#)
- [DROP TABLE](#)
- [GET_PROJECTIONS](#)
- [GET_PROJECTION_STATUS](#)
- [MARK_DESIGN_KSAFE](#)
- [Adding Nodes](#)

DROP RESOURCE POOL

Drops a user-created resource pool. All memory allocated to the pool is returned back to the GENERAL [pool](#).

Syntax

```
DROP RESOURCE POOL pool-name
```

Parameters

<i>pool-name</i>	Specifies the name of the resource pool to be dropped.
------------------	--

Privileges

Must be a superuser to drop a resource pool.

Dropping a Secondary Pool

If you try to drop a resource pool that is a secondary pool for another resource pool, Vertica returns an error. The error lists the resource pools that depend on the secondary pool you tried to drop. To drop a secondary resource pool, first set the `CASCADE TO` parameter to `DEFAULT` on the primary resource pool, and then drop the secondary pool.

For example, you can drop resource pool `rp2`, which is a secondary pool for `rp1`, as follows:

```
=> ALTER RESOURCE POOL rp1 CASCADE TO DEFAULT;  
=> DROP RESOURCE POOL rp2;
```

Transferring Resource Requests

Any requests queued against the pool are transferred to the `GENERAL` pool according to the priority of the pool compared to the `GENERAL` pool. If the pool's priority is higher than the `GENERAL` pool, the requests are placed at the head of the queue; otherwise the requests are placed at the end of the queue.

Any users who are using the pool are switched to use the `GENERAL` pool with a `NOTICE`:

```
NOTICE: Switched the following users to the General pool: username
```

`DROP RESOURCE POOL` returns an error if the user does not have permission to use the `GENERAL` pool. Existing sessions are transferred to the `GENERAL` pool regardless of whether the session's user has permission to use the `GENERAL` pool. This can result in additional user privileges if the pool being dropped is more restrictive than the `GENERAL` pool. To prevent giving users additional privileges, follow this procedure to drop restrictive pools:

1. [Revoke the permissions on the pool](#) for all users.
2. Close any sessions that had permissions on the pool.
3. Drop the resource pool.

Examples

The following command drops the resource pool that was created for the CEO:

```
=> DROP RESOURCE POOL ceo_pool;
```

See Also

- [ALTER RESOURCE POOL](#)
- [CREATE RESOURCE POOL](#)
- [Managing Workloads](#)

DROP ROLE

Removes a role from the database. Only the database superuser can drop a role.

Use the CASCADE option to drop a role that is assigned to one or more users or roles.

NOTE: You cannot use the DROP ROLE command on a role added to the Vertica database with the LDAPLink service.

Syntax

```
DROP ROLE role [CASCADE];
```

Parameters

<i>role</i>	The name of the role to drop
CASCADE	Revoke the role from users and other roles before dropping the role

Privileges

Must be a superuser to drop a role.

Examples

```
=> DROP ROLE appadmin;  
NOTICE: User bob depends on Role appadmin  
ROLLBACK: DROP ROLE failed due to dependencies  
DETAIL: Cannot drop Role appadmin because other objects depend on it  
HINT: Use DROP ROLE ... CASCADE to remove granted roles from the dependent users/roles  
=> DROP ROLE appadmin CASCADE;  
DROP ROLE
```

See Also

- [ALTER ROLE RENAME](#)
- [CREATE ROLE](#)

DROP SCHEMA

Permanently removes a schema from the database. Be sure that you want to remove the schema before you drop it, because DROP SCHEMA is an irreversible process. Use the CASCADE parameter to drop a schema containing one or more objects.

Syntax

```
DROP SCHEMA schema[,...] [ CASCADE | RESTRICT ]
```

Parameters

<i>schema</i>	Name of the schema to drop.
CASCADE	Specifies to drop the schema and all objects in it, regardless of who owns

	those objects. Caution: Objects in other schemas that depend on objects in the dropped schema—for example, user-defined functions—also are silently dropped.
RESTRICT	Drops the schema only if it is empty (default).

Privileges

Schema owner

Restrictions

- You cannot drop the PUBLIC schema.
- If a user is accessing an object within a schema that is in the process of being dropped, the schema is not deleted until the transaction completes.
- Canceling a DROP SCHEMA statement can cause unpredictable results.

Examples

The following example drops schema S1 only if it doesn't contain any objects:

```
=> DROP SCHEMA S1;
```

The following example drops schema S1 whether or not it contains objects:

```
=> DROP SCHEMA S1 CASCADE;
```

DROP SEQUENCE

Removes the specified named sequence number generator.

Syntax

```
DROP SEQUENCE [schema.]sequence-name[, ...]
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>sequence-name</i>	<p>Specifies the sequence to drop.</p>

Privileges

One of the following:

- Superuser
- Sequence or schema owner

Restrictions

- For sequences specified in a table's default expression, the default expression fails the next time you try to load data. Vertica does not check for these instances.
- `DROP SEQUENCE` does not support the `CASCADE` keyword. Sequences used in a default expression of a column cannot be dropped until all references to the sequence are removed from the default expression.

Example

The following command drops the sequence named `sequential`.

```
=> DROP SEQUENCE sequential;
```

See Also

- [ALTER SEQUENCE](#)
- [CREATE SEQUENCE](#)
- [CURRVAL](#)
- [GRANT \(Sequence\)](#)
- [NEXTVAL](#)
- [Working with Sequence Types](#)
- [Sequence Privileges](#)

DROP SUBNET

Removes a subnet from Vertica. You can use the CASCADE option to also remove the subnet from any database definition. (See [Identify the Database or Nodes Used for Import/Export](#) for more information.)

Syntax

```
DROP SUBNET subnet-name [CASCADE]
```

Parameters

The parameters are defined as follows:

<i>subnet-name</i>	The subnet you want to remove.
--------------------	--------------------------------

If you remove a subnet, be sure your database is not configured to allow export on the public subnet. (See [Identify the Database or Nodes Used for Import/Export](#) for more information.)

Privileges

Must be a superuser to drop a subnet.

Examples

This example shows how to remove a subnet.

```
=> DROP SUBNET mySubnet;
```

DROP TABLE

Removes a table and its projections. When you run `DROP TABLE`, the change is auto-committed.

Syntax

```
DROP TABLE [ IF EXISTS ] [ schema. ] table-name [, ...] [ CASCADE ]
```

Parameters

<code>IF EXISTS</code>	Specifies not to report an error if one or more of the tables to drop does not exist. This clause is useful in SQL scripts where you want to drop a table if it exists before recreating it.
<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>table-name</i>	The table to drop.
<code>CASCADE</code>	Specifies to drop all projections for this table before the table is dropped. <code>CASCADE</code> is optional if only auto-projections are associated with this table; otherwise it is required.

	This option is not valid for external tables.
--	---

Privileges

- Table owner with USAGE privilege on table's schema
- Schema owner

Requirements

- Do not cancel an executing DROP TABLE. Doing so can leave the database in an inconsistent state.
- Check that the target table is not in use, either directly or indirectly—for example, in a view.
- If you drop and restore a table that is referenced by a view, the new table must have the same name and column definitions.

Examples

See [Dropping Tables](#) in the Administrator's Guide.

See Also

- [DROP PROJECTION](#)
- [TRUNCATE TABLE](#)

DROP TEXT INDEX

Drops a text index used to perform text searches.

Syntax

```
DROP TEXT INDEX [ IF EXISTS ] [schema.]idx-table
```

Parameters

<code>[IF EXISTS]</code>	If specified, DROP TEXT INDEX does not report an error if one or more of the tables to be dropped does not exist. You can use this clause in SQL scripts where you want to drop a table, if it exists, before recreating it.
<code>schema</code>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<code>idx-table</code>	Specifies the text index name. When using more than one schema, specify the schema that contains the index in the DROP TEXT INDEX statement.

Privileges

Table owner and dbadmin with USAGE privilege on schema that contains the table or schema owner.

Behavior

When a source table is dropped that has a text index associated with it, the text index is also dropped.

Examples

The following example drops the text index `t_text_index`:

```
=> DROP TEXT INDEX t_text_index;  
DROP INDEX
```

See Also

- [Using Text Search](#)
- [CREATE TEXT INDEX](#)

DROP TRANSFORM FUNCTION

Drops a User Defined Transform Function (UDTF) from the Vertica catalog.

Syntax

```
DROP TRANSFORM FUNCTION [schema.]function-name( [ arg-List ] )
```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.mytransformfunction</pre>
<i>function-name</i>	Specifies the transform function to drop.
<i>arg-List</i>	<p>A comma-delimited list of arguments as defined for this function when it was created, specified as follows:</p> <pre>[<i>arg-name</i>] <i>arg-type</i>[, ...]</pre> <p>If multiple functions share the same name, specify the argument types of the function you wish to drop.</p> <ul style="list-style-type: none">• <i>arg-name</i> is typically a column name. You (optionally) use an <i>arg-name</i> to qualify an <i>arg-type</i>.• <i>arg-type</i> is the name of an SQL data type supported by Vertica. <p>You do not need to include an <i>arg-List</i> when dropping a polymorphic function.</p>

Privileges

One of the following:

- Superuser
- Schema or function owner

Example

The following command drops the `tokenize` UDTF in the `macros` schema:

```
=> DROP TRANSFORM FUNCTION macros.tokenize(varchar);  
DROP TRANSFORM FUNCTION
```

The following command drops the `Pagerank` polymorphic function in the `online` schema:

```
=> DROP TRANSFORM FUNCTION online.Pagerank();  
DROP TRANSFORM FUNCTION
```

See Also

[CREATE TRANSFORM FUNCTION](#)

DROP USER

Removes a name from the list of authorized database users.

NOTE: You cannot use the `DROP USER` command on a user added to the Vertica database with the `LDAPLink` service.

Syntax

```
DROP USER name [, ...] [ CASCADE ]
```

Parameters

<i>name</i>	Specifies the name or names of the user to drop.
CASCADE	[Optional] Drops all user-defined objects created by the user dropped, including schema, table and all views that reference the table, and the table's associated projections.

Privileges

Must be a superuser to drop a user.

Examples

`DROP USER <username>` fails if objects exist that were created by the user, such as schemas, and tables and their associated projections:

```
=> DROP USER user1;
NOTICE: Table T_tbd1 depends on User user1
ROLLBACK: DROP failed due to dependencies
DETAIL: Cannot drop User user1 because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too
```

`DROP USER <name> CASCADE` succeeds regardless of any pre-existing user-defined objects. The statement forcibly drops all user-defined objects, such as schemas, tables and their associated projections:

```
=> DROP USER user1 CASCADE;
```

Caution: Tables owned by the user being dropped cannot be recovered after you issue `DROP USER CASCADE`.

`DROP USER <username>` succeeds if no user-defined objects exist (no schemas, tables or projections defined by the user):

```
=> CREATE USER user2;
CREATE USER
=> DROP USER user2;
DROP USER
```

See Also

- [ALTER USER](#)
- [CREATE USER](#)

DROP VIEW

Removes the specified view. Vertica does not check for dependencies on the dropped view. After dropping a view, other views that reference it will fail.

If you drop a view and replace it with another view or table with the same name and column names, other views that reference that name use the new view. If you change the column data type in the new view, the server coerces the old data type to the new one if possible; otherwise, it returns an error.

Syntax

```
DROP VIEW name [ , ... ]
```

Parameters

<i>name</i>	Specifies the view to drop.
-------------	-----------------------------

Privileges

To drop a view, you must be the view owner and have USAGE privileges on the schema or be the schema owner.

Examples

```
=> DROP VIEW myview;
```

END

Ends the current transaction and makes all changes that occurred during the transaction permanent and visible to other users.

Syntax

```
COMMIT [ WORK | TRANSACTION ]
```

Parameters

WORK TRANSACTION	Have no effect; they are optional keywords for readability.
--------------------	---

Privileges

No special permissions required.

Notes

[COMMIT](#) is a synonym for [END](#).

Examples

This example shows how to end a transaction.

```
=> CREATE TABLE sample_table (a INT);
=> INSERT INTO sample_table (a) VALUES (1);
OUTPUT
-----
1
=> END;
COMMIT
```

See Also

- [Transactions](#)
- [Creating Transactions](#)
- [BEGIN](#)
- [ROLLBACK](#)
- [START TRANSACTION](#)

EXPLAIN

Returns a formatted description of the Vertica optimizer's plan for executing the specified statement. For detailed information, see [EXPLAIN Output Options](#) in the Administrator's Guide.

Syntax

```
EXPLAIN [/*+ ALLNODES */] [ VERBOSE ] [ JSON ] [ LOCAL ] sql-statement
```

Parameters

<code>/*+ALLNODES*/</code>	Specifies to create a query plan that assumes all nodes are active.
<code>VERBOSE</code>	Increases the level of detail in the rendered query plan.
<code>JSON</code>	Renders the query plan in JSON format.
<code>LOCAL</code>	On a multi-node database, shows the local query plans assigned to each node, which together comprise the total (global) query plan. If you omit this option, Vertica shows only the global query plan. Local query plan are shown only in DOT language source, which can be rendered in Graphviz .
<code><i>sql-statement</i></code>	A query (SELECT) statement or DML statement—for example, INSERT , UPDATE , COPY , and MERGE .

Privileges

The same privileges required by the specified statement.

Requirements

The following requirements apply to EXPLAIN's ability to produce useful information:

- Reasonably representative statistics of your data must be available. See [Collecting Statistics](#) in the Administrator's Guide for details.
- EXPLAIN produces useful output only if projections are available for the queried tables.

EXPORT TO PARQUET

Exports a table, columns from a table, or query results to files in the Parquet format. You can use an OVER() clause to partition the data before export. Partitioning data can improve query performance by enabling partition pruning; see [Improving Query Performance for Data Stored in HDFS](#).

You can export data stored in Vertica in ROS format and data from external tables.

The exported files are owned by the superuser.

EXPORT TO PARQUET returns the number of rows written.

During the export, Vertica writes files to a temporary directory in the same location as the destination and renames the directory when the export is complete. Do not attempt to use the files in the temporary directory.

Syntax

```
EXPORT TO PARQUET ( directory = 'directory'  
                    [, compression = 'compression_type' ]  
                    [, rowGroupSizeMB = 'size' ] )  
  [ OVER (over-clause ) ]  
  AS SELECT query-expression;
```

Parameters

<i>directory</i>	The destination directory for the Parquet files. The directory must not exist, and the current user must have permission to write it. The directory can be in HDFS or on the local file system.
<i>compression_type</i>	Column compression type, one of Snappy or Uncompressed. The default is Snappy.
<i>size</i>	The uncompressed size of exported row groups, in MB. The maximum value is 512 and the default is 64. The row groups in the exported files are smaller because Parquet files are compressed on write. For best performance, set <i>size</i> to be smaller than the HDFS block size.

Arguments

<i>over-clause</i>	<p>Specifies how to partition table data using PARTITION BY. Within partitions you can sort by using ORDER BY. See SQL Analytics. This clause may contain column references but not expressions.</p> <p>If you partition data, Vertica creates a Hive-style partition directory structure. See Using Partition Columns for a description of the structure.</p> <p>If you omit this clause, Vertica optimizes for maximum parallelism.</p>
<i>query-expression</i>	Specifies the data to be exported. See SELECT for the syntax.

Privileges

- SELECT privileges on the source table
- USAGE privileges on source table schema
- Write privileges for the destination directory

Notes

- This operation does not support the TIME, TIMEZ, and INTERVAL data types. Decimal precision must be ≤ 38 .
- Vertica does not convert TIMESTAMP values to UTC. To avoid problems arising from time zones, use TIMESTAMPTZ instead of TIMESTAMP.
- You must provide an alias column label for selected column targets that are expressions.
- The OVER() clause, if specified, can contain only column references, not an expression.
- This operation exports raw Flex columns as binary data.
- The exported Hive types might not be identical to the Vertica types. For example, a Vertica INT is exported as a Hive BIGINT. When defining Hive external tables to read exported data, you might have to adjust column definitions.
- If you specify a directory in the local file system that is not shared storage, Vertica distributes the files among nodes according to how the export is partitioned. To write all local files on one node, use an empty OVER() clause.
- If you specify a directory in the local file system, you must have a USER storage location.
- Vertica does not support simultaneous exports to the same directory in HDFS. The results are undefined.
- Output file names follow the pattern: [8-character hash]-[node name]-[thread_id].parquet.
- Parquet files exported to a local filesystem by any Vertica user are owned by the Vertica superuser. Parquet files exported to HDFS are owned by the Vertica user who exported the data.

Examples

The following example demonstrates exporting all columns from the T1 table in the public schema, using Snappy compression (the default).

```
=> EXPORT TO PARQUET(directory = 'hdfs:///user1/data')  
AS SELECT * FROM public.T1;
```

```
Rows Exported
-----
      87436
(1 row)
```

The following example demonstrates exporting the results of a query using more than one table.

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_by_region')
   AS SELECT sale.price, sale.date, store.region
   FROM public.sales sale
   JOIN public.vendor store ON sale.distribID = store.ID;
Rows Exported
-----
     23301
(1 row)
```

The following example demonstrates partitioning and exporting data. EXPORT TO PARQUET first partitions the data on 'b' and then, within each partition, sorts by 'd'.

```
=> EXPORT TO PARQUET(directory = 'hdfs:///user2/data')
   OVER(PARTITION BY b ORDER BY d) AS SELECT b, d FROM public.T2;

Rows Exported
-----
     120931
(1 row)
```

The following example uses an alias column label for a selected column target that is an expression.

```
=> EXPORT TO PARQUET(directory = 'hdfs:///user3/data')
   OVER(ORDER BY col1) AS SELECT col1 + col1 AS A, col2
   FROM public.T3;
Rows Exported
-----
     14336
(1 row)
```

EXPORT TO VERTICA

Exports an entire table, columns from a table, or query results to another Vertica database. Exported data is always written in [AUTO mode](#).

Important: The source database can be one major release behind the target database.

Syntax

```
EXPORT TO VERTICA database.[schema.]target-table  
... [ ( target-column[,...] ) ]  
... { AS SELECT query-expression | FROM [schema.]source-table[ ( source-column[,...] ) ] };
```

Parameters

<i>database</i>	A string containing the name of the database to receive the exported data. There must be an active connection to this database for the export to succeed.
[<i>schema</i> .] <i>target-table</i>	The table to store the exported data (schema specification is optional). This table must already exist.
<i>target-column</i>	A list of columns in the target table to store the exported data.
<i>query-expression</i>	Specifies the data to be exported. See SELECT for the syntax.
[<i>schema</i> .] <i>source-table</i>	The table that contains the data to export.
<i>source-column</i>	A list of the columns in the source table to export. If present, only these columns are exported.

Privileges

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema

Connecting to the Target Database

Before you can export data to another database, you must establish a connection to the target database with `CONNECT`. See [Exporting Data to Another Vertica Database](#) for details.

By default, `EXPORT TO VERTICA` exports data to another database over the Vertica private network. Connecting to a public network requires some configuration. For details about exporting data across a public network, see [Using Public and Private IP Networks](#).

The export operation fails if either side of the connection is a single-node cluster installed to `localhost`, or you do not specify a host name or IP address.

Source and Target Column Mapping

You can optionally name a subset of source and target columns to participate in the export operation. `EXPORT TO VERTICA` tries to match columns in the source table with corresponding columns in the destination table. If you do not supply a list of source and destination columns, `EXPORT TO VERTICA` tries to match columns in the source table with corresponding columns in the destination table. Auto-projections for the target table are similar to the projections for the source table.

The following table compares the different combinations of naming source and target columns, and the requirements that pertain to each option.

	Omit source columns	Specify source columns
Omit target columns	<p>Match all columns in source table to columns in target table.</p> <p>The number of columns in the two tables can differ, but the target table must have at least as many columns as the source table.</p>	<p>Match named source table columns to target table columns.</p> <p>The number of columns in the two tables can differ, but the target table must have at least as many columns as the number of specified source columns.</p>
Specify target columns	<p>Match source columns to the named target columns.</p> <p>The number of named target columns must equal the number of columns in the source table.</p>	<p>Match named source columns to named target columns.</p> <p>The number of source and target columns must be equal.</p>

Node Failure During EXPORT

See [Handling Node Failure During Copy/Export](#) in the Administrator's Guide.

Examples

See [Exporting Data to Another Vertica Database](#) in the Administrator's Guide.

See Also

[COPY FROM VERTICA](#)

GRANT Statements

GRANT statements, described in this section, allow you to grant privileges on database objects to specific users:

- [GRANT \(Authentication\)](#)
- [GRANT \(Database\)](#)
- [GRANT \(Library\)](#)
- [GRANT \(Model\)](#)
- [GRANT \(Procedure\)](#)
- [GRANT \(Resource Pool\)](#)
- [GRANT \(Role\)](#)
- [GRANT \(Schema\)](#)
- [GRANT \(Sequence\)](#)
- [GRANT \(Storage Location\)](#)
- [GRANT \(Table\)](#)
- [GRANT \(User Defined Extension\)](#)
- [GRANT \(View\)](#)

GRANT (Authentication)

Associates (or *grants*) an authentication record to one or more users or roles.

Syntax

```
GRANT AUTHENTICATION auth-method-name TO  
  { Public | user_or_role | user_or_role1, user_or_role2, user_or_role3,... }
```

Parameters

<i>auth_method_name</i>	Name of the authentication method you want to associate with one or more users or roles. Type: VARCHAR
<i>user_or_role, user_or_role1, user_or_role2, user_or_role3, ...</i>	Names of users or roles with whom you want to associate the authentication method. Type: VARCHAR.

Privileges

Must have DBADMIN privileges.

Examples

This example uses a GRANT AUTHENTICATION statement to associate v_ldap authentication with user jsmith:

```
=> GRANT AUTHENTICATION v_ldap TO jsmith;
```

This example uses a GRANT AUTHENTICATION statement to associate v_gss authentication to the role DBprogrammer:

```
=> CREATE ROLE DBprogrammer;  
=> GRANT AUTHENTICATION v_gss to DBprogrammer;
```

This example sets the default client authentication method to v_localpwd:

```
=> GRANT AUTHENTICATION v_localpwd TO Public;
```

See Also

- [ALTER AUTHENTICATION](#)
- [CREATE AUTHENTICATION](#)

- [DROP AUTHENTICATION](#)
- [REVOKE \(Authentication\)](#)

GRANT (Database)

Grants the right to create schemas within the database to a user or role. By default, only the superuser has the right to create a database schema.

Syntax

```
GRANT { ... { CREATE [, ...]  
... | { TEMP }  
... | ALL [ PRIVILEGES ]  
... | CONNECT } }  
... ON DATABASE database-name [, ...]  
... TO { username | rolename } [, ...]  
... [ WITH GRANT OPTION ]
```

Parameters

CREATE	Allows the user to create schemas within the specified database.
TEMP	Allows the user to create temporary tables in the database.
CONNECT	Allows the user to connect to a database.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>database-name</i>	Identifies the database in which to grant the privilege.
<i>username</i> <i>rolename</i>	Grants the privilege to the specified user or role.
WITH GRANT OPTION	Allows the recipient of the privilege to grant it to other users.

Example

The following example grants user Fred the right to create schemas on vmartdb.

```
=> GRANT CREATE ON DATABASE vmartdb TO Fred;
```

See Also

- [REVOKE \(Database\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Library)

Grants privileges on a library to a database user or role. Only the superuser can grant privileges to a library. To execute functions inside the library, a user must have separate EXECUTE privileges for those functions.

For example, when working with the Connector Framework Service you may need to grant a user usage privileges to a library to be able to set UDSession parameters. For more information see [Implementing CFS](#).

Syntax

```
GRANT { USAGE | ALL }  
  ON LIBRARY [ [ db-name.]schema.]Library-name [, ...]  
  TO { username | role | PUBLIC } [, ...]
```

Parameters

<code>{ USAGE ALL }</code>	The type of privilege to grant on the library, valid values are: <ul style="list-style-type: none">• USAGE• ALL
<code>[[<i>db-name</i>.]<i>schema-name</i>.]</code>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<code><i>Library-name</i></code>	The library on which to grant the privilege. If using more

	than one schema, you must specify the schema that where the library resides.
{ <i>username</i> <i>role</i> PUBLIC } [,...]	<p>The recipient of the privileges, valid values are:</p> <ul style="list-style-type: none"> • username—Assigns privileges to a specific user • role—Assigns privileges to a role • PUBLIC—Assigns privileges to all users and roles.

Example

The following command grants USAGE privileges on the `idolLib` library in the `v_idol` schema to `user1`:

```
=> GRANT USAGE ON LIBRARY v_idol.IdolLib TO user1;
```

See Also

- [REVOKE \(Library\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Model)

Grants access to models to a database user or role.

Syntax

```
GRANT {ALL | USAGE }
... ON MODEL [ [ db-name. ]schema. ]modelName [ , ... ]
... TO { username | role | PUBLIC }
... [ WITH GRANT OPTION ]
```

Parameters

ALL USAGE	Applies to all privileges.
-------------	----------------------------

<code>[db-name.]schema.database-name</code>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema</pre>
<code>username] role] PUBLIC</code>	The name of the user, role or group to be granted access.
<code>WITH GRANT OPTION</code>	Allows the user to grant the same privileges to other users.

Example

This example grants USAGE privileges on the mySvmClassModel model to user1:

```
=> GRANT USAGE ON MODEL mySvmClassModel TO user1;
```

See Also

- [REVOKE \(Model\)](#)
- [Managing Model Security](#)

GRANT (Procedure)

Grants privileges on a procedure to a database user or role. Only the superuser can grant privileges to a procedure. To grant privileges to a schema containing the procedure, users must have USAGE privileges. See [GRANT \(Schema\)](#).

Important: External procedures that you create with [CREATE PROCEDURE](#) are always run with Linux dbadmin privileges. If a dbadmin or pseudosuperuser grants a non-dbadmin permission to run a procedure using [GRANT \(Procedure\)](#), be aware that the non-dbadmin user runs the procedure with full Linux dbadmin privileges.

Syntax

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }  
  ON PROCEDURE [ [schema.]procedure-name [, ...]  
  ( [ argname ] argtype [,... ] )  
  TO { username | role | PUBLIC } [, ...]
```

Parameters

{ EXECUTE ALL }	The type of privilege to grant the procedure. Either EXECUTE or ALL are applicable privileges to grant. When using more than one schema, specify the schema that contains the procedure.
PRIVILEGES	[Optional] For SQL standard compatibility and is ignored.
<i>schema-name</i>	Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>procedure-name</i>	The SQL or User Defined procedure on which to grant the privilege. If using more than one schema, you must specify the schema that contains the procedure.
<i>argname</i>	The optional argument name for the procedure.
<i>argtype</i>	The required argument data type or types of the procedure.
{ <i>username</i> <i>role</i> PUBLIC }[,...]	The recipient of the procedure privileges, which can be one or more users, one or more roles, or all users and roles (PUBLIC). <ul style="list-style-type: none"> <i>username</i>—Indicates a specific user <i>role</i>—Specifies a particular role PUBLIC—Indicates that all users and roles have granted privileges to the procedure.

Example

The following command grants EXECUTE privileges on the tokenize procedure to users Bob and Jules, and to the Operator role:

```
=> GRANT EXECUTE ON PROCEDURE tokenize(varchar) TO Bob, Jules, Operator;
```

See Also

- [REVOKE \(Procedure\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Resource Pool)

Grants privileges on one or more resource pools to a database user or role. Once granted usage rights, users can switch to using the resource pool with [ALTER USER \(username\)](#) or with [SET SESSION RESOURCE POOL](#).

Syntax

```
GRANT USAGE
  ON RESOURCE POOL resource-pool [, ...]
  TO { username | role | PUBLIC } [, ...]
```

Parameters

<i>resource-pool</i>	The resource pools on which to grant the privilege.
{ <i>username</i> <i>role</i> PUBLIC } [,...]	The recipient of the procedure privileges, which can be one or more users, one or more roles, or all users and roles (PUBLIC). <ul style="list-style-type: none">• <i>username</i>—Indicates one or more user names.• <i>role</i>—Indicates one or more roles.• PUBLIC—Indicates that all users and roles have granted privileges to the procedure.

Examples

This examples shows how to grant user Joe usage on resource pool Joe_pool1.

```
=> CREATE USER Joe;  
CREATE USER  
  
=> CREATE RESOURCE POOL Joe_pool;  
CREATE RESOURCE POOL  
  
=> GRANT USAGE ON RESOURCE POOL Joe_pool TO Joe;  
GRANT PRIVILEGE
```

See Also

- [REVOKE \(Resource Pool\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Role)

Adds a predefined role to users or other roles. Granting a role does not activate the role automatically; the user must [enable it](#) using the [SET ROLE](#) command.

Granting a privilege to a role immediately affects active user sessions. When you grant a new privilege, it becomes immediately available to every user with the role active.

Syntax

```
GRANT role [,...] TO { user | role | PUBLIC } [, ...]  
... [ WITH ADMIN OPTION ];
```

Parameters

<i>role</i> [,...]	The name of one or more roles to be granted to users or roles
<i>user</i> <i>role</i> PUBLIC	The name of a user or other role to be granted the role. If the keyword PUBLIC is supplied, then all users have access to the role.
WITH ADMIN OPTION	Grants users and roles administrative privileges for the role. They are able to grant the role to and revoke the role from other users or roles.

Notes

Vertica will return a NOTICE if you grant a role with or without admin option, to a grantee who has already been granted that role. For example:

```
=> GRANT commentor to Bob;  
NOTICE 4622: Role "commentor" was already granted to user "Bob"
```

Creating Roles

These examples create three roles, appdata, applogs, and appadmin, and grant one of the roles to a user, bob:

```
=> CREATE ROLE appdata;  
CREATE ROLE  
=> CREATE ROLE applogs;  
CREATE ROLE  
=> CREATE ROLE appadmin;  
CREATE ROLE  
=> GRANT appdata TO bob;  
GRANT ROLE
```

Activating a Role

After granting a role to a user, the role must be activated. You can activate a role on a session basis, or as part of the user's login.

To activate a role for a user's session:

```
=> CREATE ROLE appdata;  
CREATE ROLE  
=> GRANT appdata TO bob;  
GRANT ROLE  
=> SET ROLE appdata;  
SET ROLE
```

To activate a role as part of the user's login:

```
=> CREATE ROLE appdata;  
CREATE ROLE  
=> GRANT appdata TO bob;  
GRANT ROLE  
=> ALTER USER bob DEFAULT ROLE appdata;  
ALTER USER
```

Granting One Role To Another

Grant two roles to another role:

```
=> GRANT appdata, applogs TO appadmin;  
-- grant to other roles  
GRANT ROLE
```

Now, any privileges assigned to either appdata or applogs are automatically assigned to appadmin as well.

Checking for Circular References

When you grant one role to another role, Vertica combines the newly granted role's permissions with the existing role's permissions. Vertica also checks for circular references when you grant one role to another. The GRANT ROLE function fails with an error if a circular reference is found.

```
=> GRANT appadmin TO appdata;  
WARNING: Circular assignation of roles is not allowed  
HINT: Cannot grant appadmin to appdata  
GRANT ROLE
```

Granting Administrative Privileges

A superuser can assign a user or role administrative access to a role by supplying the optional WITH ADMIN OPTION argument to the [GRANT](#) statement. Administrative access allows the user to grant and revoke access to the role for other users (including granting them administrative access). Giving users the ability to grant roles lets a superuser delegate role administration to other users.

Note: A user with a DBADMIN role must have the ADMIN OPTION enabled to be able to grant a DBADMIN role to another user.

As with all user privilege models, database superusers should be cautious when granting any user a role with administrative privileges. For example, if the database superuser grants two users a role with administrative privileges, both users can revoke the role of the other user. This example shows granting the appadmin role (with administrative privileges) to users bob and alice. After each user has been granted the appadmin role, either use can connect as the other will full privileges.

```
=> GRANT appadmin TO bob, alice WITH ADMIN OPTION;
GRANT ROLE
=> \connect - bob
You are now connected as user "bob".
=> REVOKE appadmin FROM alice;
REVOKE ROLE
```

See Also

- [REVOKE \(Role\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Schema)

Grants privileges on a schema to a database user or role.

Note: If a schema was created with Inherited Privileges enabled, any privileges you grant the schema are inherited by all the objects in the table. Otherwise, you need to grant privileges on each object in the table. For more information see [CREATE SCHEMA](#)

New users do not have access to schema PUBLIC by default. You must grant USAGE on the PUBLIC schema to all users you create.

Syntax

```
GRANT { ... { CREATE | USAGE } [ , ... ] | ALL [ PRIVILEGES ] }
... | { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [ ,... ] ... }
... ON SCHEMA [db-name.]schema [ , ... ]
... TO { username | role | PUBLIC } [ , ... ]
... [ WITH GRANT OPTION ]
```

Parameters

CREATE	Grants the user read access to the schema and the right to create tables and views within the schema.
USAGE	Grants the user access to the objects contained within the schema. This allows the user to look up objects within the schema. Note that the user must also be granted access to the individual objects. See the GRANT TABLE and GRANT VIEW statements.

SELECT	With Inherited Privileges enabled on the schema, grants the user SELECT privileges on any column of any table in the schema. See Inherit Privileges on a Schema .
INSERT	With Inherited Privileges enabled on the schema, grants the user privileges to INSERT tuples into tables in the schema and use the COPY command to load data into the tables. See Inherit Privileges on a Schema .
UPDATE	With Inherited Privileges enabled on the schema, grants the user privileges to UPDATE tuples in a schema table. See Inherit Privileges on a Schema .
DELETE	With Inherited Privileges enabled on the schema, grants the user privileges to DELETE rows from a schema table. See Inherit Privileges on a Schema .
REFERENCES	With Inherited Privileges enabled on the schema, grants the ability to create a foreign key constraint. You must have this privilege on both the referencing and referenced tables. You also need USAGE on the schema that contains the table. See Inherit Privileges on a Schema .
TRUNCATE	With Inherited Privileges enabled on the schema, grants the user TRUNCATE privileges on rows from a schema table. See Inherit Privileges on a Schema .
ALL	Grants the user CREATE and USAGE privileges on the schema.
PRIVILEGES	Used for SQL standard compatibility.
[db-name.]	[Optional] Specifies the current database name. Using a database name prefix is optional, and does not affect the command in any way. You must be connected to the specified database.
<i>schema</i>	Identifies the schema to which you are granting privileges.
<i>username</i>	Grants the privilege to a specific user.
<i>role</i>	Grants the privilege to a specific role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the recipient of the privilege to grant it to other users.

Examples

This example shows how to grant user Joe usage on schema `online_sales`.

```
=> CREATE USER Joe;  
CREATE USER  
  
=> GRANT USAGE ON SCHEMA online_sales TO Joe;  
GRANT PRIVILEGE
```

See Also

- [REVOKE \(Schema\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Sequence)

Grants privileges on a sequence generator to a user or role. Optionally grants privileges on all sequences within one or more schemas.

Syntax

```
GRANT { SELECT | ALL [ PRIVILEGES ] }... ON SEQUENCE [schema.]sequence-name[,...]  
... | ON ALL SEQUENCES IN SCHEMA schema-name[,...]  
... TO { username | role | PUBLIC }[,...]  
... [ WITH GRANT OPTION ]
```

Parameters

SELECT	Allows the right to use both the CURRVAL and NEXTVAL functions on the specified sequence.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[<i>db-name.</i>] <i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>

<i>sequence-name</i>	Specifies the sequence on which to grant the privileges. When using more than one schema, specify the schema that contains the sequence on which to grant privileges.
ON ALL SEQUENCES IN SCHEMA	Grants privileges on all sequences within one or more schemas to a user and/or role.
<i>username</i>	Grants the privilege to the specified user.
<i>role</i>	Grants the privilege to the specified role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the user to grant the same privileges to other users.

Privileges

[USAGE on the schema](#) that contains the sequence

Examples

This example shows how to grant user Joe all privileges on sequence my_seq.

```
=> CREATE SEQUENCE my_seq START 100;  
CREATE SEQUENCE  
  
=> GRANT ALL PRIVILEGES ON SEQUENCE my_seq TO Joe;  
GRANT PRIVILEGE
```

See Also

- [REVOKE \(Sequence\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Storage Location)

Grants privileges to non-superusers or roles to read from or write to a Vertica storage location. First, a superuser [creates](#) a special class of storage location with the USER keyword through the usage parameter. Creating a storage location with a USER type specifies that the location can

be made accessible to non-dbadmin users. The superuser must then grant users or roles the appropriate privileges through the GRANT (Storage Location) statement.

Note: GRANT/REVOKE (Storage Location) statements are applicable only to 'USER' storage locations. If the storage location is dropped, all privileges are revoked automatically.

Syntax

```
GRANT { READ | WRITE | ALL [ PRIVILEGES ] }
... ON LOCATION 'path' [ ON node ]
... TO { username | role | PUBLIC } [, ...]
... [ WITH GRANT OPTION ]
```

Parameters

READ	Permits the grantee to copy data from files in the storage location into a table.
WRITE	Permits the grantee to export data from the database to the storage location. With WRITE privileges, grantees can also save COPY statement rejected data and exceptions files to the storage location.
ALL	Grants all available privileges to the grantee for the storage location.
PRIVILEGES	[Optional] For SQL standard compatibility and is ignored.
ON LOCATION ' <i>path</i> ' [ON <i>node</i>]	<ul style="list-style-type: none"> <i>path</i> — Specifies the path name mount point of the storage location <i>node</i> — [Optional] Grants access to the storage location residing on the node. If you leave this blank, <i>node</i> defaults to all nodes on the specified path in that cluster. If a path exists for only some nodes, the entire grant rolls back, even on the nodes that reside in the path.
{ <i>username</i> <i>role</i> PUBLIC } [,...]	<p>Specifies the privilege grantee, which can be one or more users, one or more roles, or all users (PUBLIC).</p> <ul style="list-style-type: none"> <i>username</i> – A specific user

	<ul style="list-style-type: none">• <i>role</i> – A particular role• PUBLIC – Grants the specified privileges to all users and roles.
WITH GRANT OPTION	[Optional] Allows the grantee to grant the same privileges to others.

Notes

Only a superuser can add, alter, retire, drop, and restore a location. The superuser can grant only READ and/or WRITE access privileges to storage locations for other users or roles.

Examples

In the following series of commands, a superuser creates a new storage location and grants it to user Bob:

```
=> CREATE LOCATION '/home/dbadmin/UserStorage/BobStore' NODE 'v_mcdb_node0007' USAGE 'USER';  
CREATE LOCATION
```

Now the superuser grants a user named Bob all available privileges to the /BobStore location:

```
=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' TO Bob;  
GRANT PRIVILEGE
```

Revoke all storage location privileges from Bob:

```
=> REVOKE ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' FROM Bob;  
REVOKE PRIVILEGE
```

Grant privileges to Bob on the BobStore location again, specifying a node:

```
=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' ON v_mcdb_node0007 TO Bob;  
GRANT PRIVILEGE
```

Revoke all storage location privileges from Bob:

```
=> REVOKE ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' ON v_mcdb_node0007 FROM Bob;  
REVOKE PRIVILEGE
```

See Also

- [Storage Management Functions](#)
- [REVOKE \(Storage Location\)](#)
- [Granting and Revoking Privileges](#)

GRANT (Table)

Grants privileges on a table to a user or role. Optionally grants privileges on all tables within one or more schemas.

Note: Granting privileges on all tables in a schema also includes privileges on all views in the same schema.

Syntax

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [ , ... ] ... | ALL [ PRIVILEGES ] }  
... ON [ TABLE ] [ [ db-name. ] schema. ] tablename [ , ... ]  
... | ON ALL TABLES IN SCHEMA schema-name [ , ... ]  
... TO { username | role | PUBLIC } [ , ... ]  
... [ WITH GRANT OPTION ]
```

Parameters

SELECT	Allows the user to SELECT from any column of the specified table.
INSERT	Allows the user to INSERT tuples into the specified table and to use the COPY command to load the table. Note: COPY FROM STDIN is allowed to any user granted the INSERT privilege, while COPY FROM <file> is an admin-only operation.
UPDATE	Allows the user to UPDATE tuples in the specified table.
DELETE	Allows the user to DELETE a row from the specified table.

REFERENCES	You must have this privilege on both the referencing and referenced tables to create a foreign key constraint. You also need USAGE on the schema that contains the table.
TRUNCATE	Grants TRUNCATE TABLE privileges to non-owners of a table. It also grants the non-owner of the table the following partition functions: <ul style="list-style-type: none"> • DROP_PARTITION • SWAP_PARTITIONS_BETWEEN_TABLES • MOVE_PARTITIONS_TO_TABLE
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>[db-name.]schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>tableName</i>	Specifies the table on which to grant the privileges.
ON ALL TABLES IN SCHEMA	Grants privileges on all tables (and by default all views) within one or more schemas to a user and/or role.
<i>username</i>	Grants the privilege to the specified user.
<i>role</i>	Grants the privilege to the specified role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Allows the user to grant the same privileges to other users.

Notes

- The user must also be granted USAGE on the schema that contains the table. See [GRANT \(Schema\)](#).
- To use the [DELETE](#) or [UPDATE](#) commands with a [WHERE Clause](#), a user must have both SELECT and UPDATE and DELETE privileges on the table.

- The user can be granted privileges on a global temporary table, but not a local temporary table.

Examples

This example shows how to grant user Joe all privileges on table `customer_table`.

```
=> CREATE USER Joe;  
CREATE USER  
  
=> GRANT ALL PRIVILEGES ON TABLE customer_dimension TO Joe;  
GRANT PRIVILEGE
```

See Also

- [REVOKE \(Table\)](#)
- [Granting and Revoking Privileges](#)

GRANT (User Defined Extension)

Grants privileges on a user-defined extension (UDx) to a database user or role. Optionally grants all privileges on the user-defined extension within one or more schemas. You can grant privileges on the following user-defined extension types:

- User Defined Functions (UDF)
 - User Defined SQL Functions
 - User Defined Scalar Functions (UDSF)
 - User Defined Transform Functions (UDTF)
 - User Defined Aggregate Functions (UDAF)
 - User Defined Analytic Functions (UDAnF)
- User Defined Load Functions (UDL)
 - UDL Filter

- UDL Parser
- UDL Source

Syntax

```
GRANT { EXECUTE | ALL }
... ON FUNCTION [ schema.]function-name [, ...]
... | ON AGGREGATE FUNCTION [ schema.]function-name [, ...]
... | ON ANALYTIC FUNCTION [ schema.]function-name [, ...]
... | ON TRANSFORM FUNCTION [schema.]function-name [, ...]
... | ON FILTER [ schema.]filter-name [, ...]
... | ON PARSER [ schema.]parser-name [, ...]
... | ON SOURCE [ schema.]source-name [, ...]
... | ON ALL FUNCTIONS IN SCHEMA schema-name [, ...]
... ( [ argname ] argtype [, ...] )
... TO { username | role | PUBLIC } [, ...]
```

Parameters

<p>{ EXECUTE ALL }</p>	<p>The type of privilege to grant the UDX:</p> <ul style="list-style-type: none"> • EXECUTE grants permission to call a user-defined extension. • ALL grants all privileges on the user-defined extension
<p><i>schema</i></p>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre style="background-color: #f0f0f0; padding: 5px;">myschema.thisDBObject</pre>
<p><i>function-name</i> <i>filter-name</i> <i>parser-name</i> <i>source-name</i></p>	<p>The name of the UDX on which to grant the privilege. If you use more than one schema, you must specify the schema that contains the UDX, as noted in the previous row.</p>
<p>ON ALL FUNCTIONS IN SCHEMA</p>	<p>Grants privileges on all UDX's within one or more schemas to a user, role, or all users and roles.</p>
<p><i>argname</i></p>	<p>[Optional] The argument name for the UDX. When you GRANT, REVOKE, or DROP privileges for a</p>

	polymorphic function, you must include an argument with the command.
<i>argtype</i>	The argument data type of the UDx.
{ <i>username</i> <i>role</i> PUBLIC } [,...]	<p>The recipient of the UDx privileges, which can be one or more users, one or more roles, or all users and roles (PUBLIC).</p> <ul style="list-style-type: none"> • <i>username</i> - Indicates a specific user • <i>role</i> - Specifies a particular role • PUBLIC - Indicates that all users and roles have granted privileges to the UDx.

Privileges

Only a superuser and owner can grant privileges on a UDx. To grant privileges to a specific schema UDx or to all UDx's within one or more schemas, grantees must have USAGE privileges on the schema. See [GRANT \(Schema\)](#).

Examples

The following command grants EXECUTE privileges on the `myzeroifnull` SQL function to users Bob and Jules, and to the Operator role. The function takes one integer argument:

```
=> GRANT EXECUTE ON FUNCTION myzeroifnull (x INT) TO Bob, Jules, Operator;
```

The following command grants EXECUTE privileges on all functions in the `zero-schema` schema to user Bob:

```
=> GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA zero-schema TO Bob;
```

The following command grants EXECUTE privileges on the `tokenize` transform function to user Bob and to the Operator role:

```
=> GRANT EXECUTE ON TRANSFORM FUNCTION tokenize(VARCHAR) TO Bob, Operator;
```

The following command grants EXECUTE privileges on the `ExampleSource()` source to user Alice.

```
=> CREATE USER Alice;  
=> GRANT USAGE ON SCHEMA hdfs TO Alice;  
=> GRANT EXECUTE ON SOURCE ExampleSource() TO Alice;
```

The next command grants ALL privileges on the ExampleSource() source to user Alice:

```
=> GRANT ALL ON SOURCE ExampleSource() TO Alice;
```

The following command grants ALL privileges on the polymorphic function Pagerank to the dbadmin role:

```
=> GRANT ALL ON TRANSFORM FUNCTION Pagerank(z varchar) to dbadmin;
```

See Also

- [REVOKE \(User Defined Extension\)](#)
- [Granting and Revoking Privileges](#) in the Administrator's Guide
- [Developing User-Defined Extensions \(UDxs\)](#) in Extending Vertica

GRANT (View)

Grants privileges on a view to a database user or role.

Syntax

```
GRANT ... { SELECT | ALL [ PRIVILEGES ] }  
... ON [ [ db-name.]schema.]viewname [, ...]  
... TO { username | role | PUBLIC } [, ...]  
... [ WITH GRANT OPTION ]
```

Parameters

SELECT	Grants a user or role SELECT operations to a view and any resources referenced within it.
ALL	Grants a user or role all privileges to a view, and any resources referenced within it.

PRIVILEGES	[Optional] For SQL standard compatibility and is ignored.
[<i>db-name.</i>] <i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>viewname</i>	Specifies the view on which to grant the privileges. When using more than one schema, specify the schema that contains the view, as noted above.
<i>username</i>	Grants the privilege to the specified user.
<i>role</i>	Grants the privilege to the specified role.
PUBLIC	Grants the privilege to all users.
WITH GRANT OPTION	Permits the user to grant the same privileges to other users.

Examples

This example shows how to grant user Joe all privileges on view ship.

```
=> CREATE VIEW ship AS SELECT * FROM public.shipping_dimension;  
CREATE VIEW  
=> GRANT ALL PRIVILEGES ON ship TO Joe;  
GRANT PRIVILEGE
```

See Also

- [REVOKE \(View\)](#)
- [Granting and Revoking Privileges](#)
- [Using Views](#)
- [Privileges Required for Common Database Operations](#)

INSERT

Inserts values into all projections of the specified table. You must insert one complete tuple at a time. By default, INSERT first uses WOS. When WOS is full, the inserted tuple overflows to ROS.

If no projections are associated with the target table, Vertica creates a superprojection to store the inserted values.

INSERT works for flex tables as well as regular database tables.

Syntax

```
INSERT [ /*+ hint[, hint] */ ] INTO [schema.]table-name
... [ ( column-list ) ]
... { DEFAULT VALUES | VALUES ( values-list ) | SELECT query-expression }
```

Parameters

<pre>/*+ hint [, hint] */</pre>	<p>One or both of the following hints:</p> <ul style="list-style-type: none"> • A load hint, one of the following: AUTO, DIRECT, or TRICKLE • LABEL
<p><i>schema</i></p>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<p><i>table-name</i></p>	<p>The target table. You cannot invoke INSERT on a projection. This can be a flex table.</p>
<p><i>column-list</i></p>	<p>A comma-delimited list of one or more target columns in this table, listed in any order. VALUES clause values are mapped to columns in the same order. If you omit this list, Vertica maps VALUES clause values to columns according to column order in the table definition.</p> <p>A list of target columns is invalid with DEFAULT VALUES.</p>

DEFAULT VALUES	<p>Fills all columns with their default values as specified in the table definition.</p> <p>You cannot specify a list of target columns with this option.</p>
VALUES (<i>values-list</i>)	<p>A comma-delimited list of one or more values to insert in the target columns, where each value is one of the following:</p> <ul style="list-style-type: none"> • <i>expression</i> resolves to a value to insert in the target column. The expression must not nest other expressions or include Vertica meta-functions. • DEFAULT inserts the default value as specified in the table definition. <p>If no value is supplied for a column, Vertica implicitly adds a DEFAULT value, if defined. Otherwise Vertica inserts a NULL value. If the column is defined as NOT NULL, INSERT returns an error.</p>
SELECT <i>query-expression</i>	<p>Specifies a query that returns the rows to insert. Isolation level applies only to the SELECT clauses and works like any query.</p>

Privileges

- Table owner or user with GRANT OPTION is grantor
- INSERT privilege on table
- USAGE privilege on schema that contains the table

Restrictions

- Vertica does not support subqueries as the target of an INSERT statement.
- If any primary key, unique key, or check constraints are enabled for automatic enforcement, Vertica enforces those constraints when you insert values into a table. If a violation occurs, Vertica rolls back the SQL statement and returns an error. This behavior occurs for INSERT, UPDATE, COPY, and MERGE SQL statements. Note that automatic constraint enforcement requires that you have the SELECT privilege on the table containing the constraint.

Examples

```
=> INSERT INTO t1 VALUES (101, 102, 103, 104);
=> INSERT INTO customer VALUES (10, 'male', 'DPR', 'MA', 35);
=> INSERT INTO retail.t1 (C0, C1) VALUES (1, 1001);
=> INSERT INTO films SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';
```

Vertica does not support subqueries or nested expressions as the target of an INSERT statement. For example, the following query returns an error message:

```
=> INSERT INTO t1 (col1, col2) VALUES ('abc', (SELECT mycolumn FROM mytable));
ERROR 4821: Subqueries not allowed in target of insert
```

You can rewrite the above query as follows:

```
=> INSERT INTO t1 (col1, col2) (SELECT 'abc', mycolumn FROM mytable);
OUTPUT
-----
      0
(1 row)
```

The following example shows how to use INSERT . . . VALUES with flex tables:

```
=> CREATE FLEX TABLE flex1();
CREATE TABLE
=> INSERT INTO flex1(a,b) VALUES (1, 'x');
OUTPUT
-----
      1
(1 row)
=> SELECT MapToString(__raw__) FROM flex1;
      MapToString
-----
{
"a" : "1",
"b" : "x"
}
(1 row)
```

The next example shows how to use INSERT . . . SELECT with flex tables:

```
=> CREATE FLEX TABLE flex2();
CREATE TABLE
=> INSERT INTO flex2(a, b) SELECT a, b, '2016-08-10 11:10' c,      'Hello' d, 3.1415 e, f from flex1;
OUTPUT
-----
      1
(1 row)
=> SELECT MapToString(__raw__) FROM flex2;
      MapToString
-----
```

```
{
  "a" : "1",
  "b" : "x",
  "c" : "2016-08-10",
  "d" : "Hello",
  "e" : 3.1415,
  "f" : null
}
(1 row)
```

MERGE

Performs update and insert operations on a target table based on the results of a join with another data set, such as a table or view. The join can match a source row with only one target row; otherwise, Vertica returns an error.

A MERGE operation uses WOS by default. If WOS fills up, data overflows to ROS.

For detailed information, see [Merging Table Data](#).

Syntax

```
MERGE [ /*+ hint[, hint] */ ]
... INTO [schema.]target-table [ [AS] alias ]
... USING source-dataset
... ON join-condition
... matching-clause[ matching-clause ]
```

Returns

Number of target table rows updated or inserted

Parameters

/+ hint[, hint]*/*

One or both of the following hints:

- A load hint, one of the following: [AUTO](#), [DIRECT](#), or [TRICKLE](#)
- [LABEL](#)

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>target-table</i>	<p>The table on which to perform update and insert operations. MERGE takes an X (exclusive) lock on the target table during the operation.</p>
<i>source-dataset</i>	<p>The data to join to <i>target-table</i>, one of the following:</p> <ul style="list-style-type: none"> • <code>[schema.]table [[AS] alias]</code> • <code>[schema.]view [[AS] alias]</code> • <code>(subquery) sq-alias</code> <p>The specified data set typically supplies the data used to update the target table and populate new rows. You can specify an external table.</p>
ON <i>join-condition</i>	<p>The conditions on which to join the target table and source data set.</p> <p>Tip: The Vertica query optimizer can create an optimized query plan for a MERGE statement only if the target table join column has a unique or primary key constraint. For details, see MERGE Optimization in the Administrator's Guide.</p>
<i>matching-clause</i>	<p>One of the following clauses:</p> <ul style="list-style-type: none"> • WHEN MATCHED THEN UPDATE • WHEN NOT MATCHED THEN INSERT <p>MERGE supports one instance of each clause, and must include at least one.</p>
WHEN MATCHED THEN UPDATE	<p>For each <i>target-table</i> row that is joined (matched) to <i>source-dataset</i>, specifies to</p>

	<p>update one or more columns:</p> <pre>WHEN MATCHED [AND <i>update-filter</i>] THEN UPDATE SET { <i>target-column</i> = <i>expression</i> }[,...]</pre> <p><i>update-filter</i> optionally filters the set of matching rows. The update filter can specify any number of conditions. Vertica evaluates each matching row against this filter, and updates only the rows that evaluate to true. For details, see Update and Insert Filters in the Administrator's Guide.</p> <p>The following requirements apply:</p> <ul style="list-style-type: none">• A MERGE statement can contain only one WHEN MATCHED clause.• <i>target-column</i> can only specify a column name in the target table. It cannot be qualified with a table name. <p>For details, see Merging Table Data in the Administrator's Guide.</p>
<p>WHEN NOT MATCHED THEN INSERT</p>	<p>For each <i>source-dataset</i> row that is not joined (not matched) to <i>target-table</i>, specifies to:</p> <ul style="list-style-type: none">• Insert a new row into <i>target-table</i>.• Populate each new row with the values specified in <i>values-list</i>. <pre>WHEN NOT MATCHED [AND <i>insert-filter</i>] THEN INSERT [(<i>column-list</i>)] VALUES (<i>values-list</i>)</pre> <p><i>column-list</i> is a comma-delimited list of one or more target columns in the target table, listed in any order. MERGE maps <i>column-list</i> columns to <i>values-list</i> values in the same order, and each column-value pair must be compatible. If you omit <i>column-list</i>, Vertica maps <i>values-list</i> values to columns according to column order in the table definition.</p>

insert-filter optionally filters the set of non-matching rows. The insert filter can specify any number of conditions. Vertica evaluates each non-matching source row against this filter. For each row that evaluates to true, Vertica inserts a new row in the target table. For details, see [Update and Insert Filters](#) in the Administrator's Guide.

The following requirements apply:

- A MERGE statement can contain only one WHEN NOT MATCHED clause.
- *column-list* can only specify column names in the target table. It cannot be qualified with a table name.
- Insert filter conditions can only reference the source data. If any condition references the target table, Vertica returns an error.

For details, see [Merging Table Data](#) in the Administrator's Guide.

Privileges

MERGE requires the following privileges:

- SELECT permissions on the source data and INSERT, UPDATE, and DELETE permissions on the target table.
- Automatic constraint enforcement requires SELECT permissions on the table containing the constraint.
- SELECT permissions on the target table if the condition in the syntax reads data from the target table. The following example grants user1 access to target table t2:

For example, the following GRANT statement grants user1 access to target table t2. This allows user1 to run the MERGE statement that follows:

```
=> GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE t2 to user1;  
GRANT PRIVILEGE
```

```
=>\c - user1
You are now connected as user "user1".

=> MERGE INTO t2 USING t1 ON t1.a = t2.a
WHEN MATCHED THEN UPDATE SET b = t1.b
WHEN NOT MATCHED THEN INSERT (a, b) VALUES (t1.a, t1.b);
```

Improving MERGE Performance

You can improve MERGE performance in several ways:

- [Design projections for optimal MERGE performance.](#)
- [Facilitate creation of optimized query plans.](#)
- Use a source data set that is smaller than the target table.

For details, see [MERGE Optimization](#) in the Administrator's Guide.

Constraint Enforcement

MERGE respects all enforced constraints in the target table. If the merge operation attempts to copy values that violate those constraints, MERGE returns with an error and rolls back the merge operation.

Caution: If you run MERGE multiple times using the same target and source table, each iteration is liable to introduce duplicate values into the target columns and return with an error.

Columns Prohibited from Merge

The following columns cannot be specified in a merge operation; attempts to do so return with an error:

- [Identity/auto-increment](#) columns, or columns whose default value is set to a [named sequence](#).
- Vmap columns such as `__raw__` in flex tables.

Examples

See [Basic MERGE Example](#) in the Administrator's Guide.

PROFILE

Profiles a single SQL statement.

Syntax

```
PROFILE { sql-statement }
```

Parameters

<i>sql-statement</i>	A query (SELECT) statement or DML statement--for example, you can profile INSERT , UPDATE , COPY , and MERGE .
----------------------	--

Output

Writes profile summary to stderr, saves details to system catalog [V_MONITOR.EXECUTION_ENGINE_PROFILES](#).

Privileges

The same privileges required to run the profiled statement

Description

PROFILE generates detailed information about how the target statement executes, and saves that information in the system catalog [V_MONITOR.EXECUTION_ENGINE_PROFILES](#). Query output is preceded by a profile summary: profile identifiers `transaction_id` and `statement_id`, initiator memory for the query, and total memory required. For example:

```
=> PROFILE SELECT customer_name, annual_income FROM public.customer_dimension WHERE (customer_gender,
annual_income) IN (SELECT customer_gender, MAX(annual_income) FROM public.customer_dimension GROUP BY
customer_gender);
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996274683334 and
statement_id=7;
NOTICE 3557: Initiator memory for query: [on pool general: 708421 KB, minimum: 554324 KB]
NOTICE 5077: Total memory required by query: [708421 KB]
  customer_name | annual_income
-----|-----
Emily G. Vogel | 999998
James M. McNulty | 999979
(2 rows)
```

Use profile identifiers to query the table for profile information on a given query.

See Also

[Profiling Single Statements](#)

RELEASE SAVEPOINT

Destroys a savepoint without undoing the effects of commands executed after the savepoint was established.

Syntax

```
RELEASE [ SAVEPOINT ] savepoint_name
```

Parameters

<i>savepoint_name</i>	Specifies the name of the savepoint to destroy.
-----------------------	---

Privileges

No special permissions required.

Notes

Once destroyed, the savepoint is unavailable as a rollback point.

Example

The following example establishes and then destroys a savepoint called `my_savepoint`. The values 101 and 102 are both inserted at commit.

```
=> INSERT INTO product_key VALUES (101);  
=> SAVEPOINT my_savepoint;  
=> INSERT INTO product_key VALUES (102);  
=> RELEASE SAVEPOINT my_savepoint;  
=> COMMIT;
```

See Also

- [SAVEPOINT](#)
- [ROLLBACK TO SAVEPOINT](#)

REVOKE Statements

The following functions allow you to revoke privileges on database objects for specified users:

- [REVOKE \(Authentication\)](#)
- [REVOKE \(Database\)](#)
- [REVOKE \(Library\)](#)
- [REVOKE \(Model\)](#)
- [REVOKE \(Procedure\)](#)
- [REVOKE \(Resource Pool\)](#)
- [REVOKE \(Role\)](#)
- [REVOKE \(Schema\)](#)
- [REVOKE \(Sequence\)](#)
- [REVOKE \(Storage Location\)](#)
- [REVOKE \(Table\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [REVOKE \(View\)](#)

REVOKE (Authentication)

Revokes an authentication method that you have granted to (associated with) one or more users or roles.

Syntax

```
REVOKE AUTHENTICATION auth-method-name FROM  
  { Public | user_or_role | user_or_role1, user_or_role2, user_or_role3,... }
```

Parameters

<i>auth_method_name</i>	Name of the authentication method which you want to revoke from one or more users. Type: VARCHAR
<i>user_or_role, user_or_role1, user_or_role2, user_or_role3, ...</i>	Names of users or user roles from whom you want to revoke the authentication method. Type: VARCHAR

Privileges

Must have DBADMIN privileges.

Examples

This example revokes `v_ldap` authentication from user `jsmith`:

```
=> REVOKE AUTHENTICATION v_ldap FROM jsmith;
```

This example revokes `v_gss` authentication from the role `DBprogrammer`:

```
=> REVOKE AUTHENTICATION v_gss FROM DBprogrammer;
```

This example removes `localpwd` as the default client authentication method:

```
=> REVOKE AUTHENTICATION localpwd from Public;
```

See Also

- [ALTER AUTHENTICATION](#)
- [CREATE AUTHENTICATION](#)

- [DROP AUTHENTICATION](#)
- [GRANT \(Authentication\)](#)

REVOKE (Database)

Revokes the right for the specified user or role to create schemas in the specified database.

Syntax

```
REVOKE [ GRANT OPTION FOR ]  
... { CREATE | TEMP [ ,... ] }  
... | CONNECT  
... | ALL [ PRIVILEGES ] }  
... ON DATABASE database-name [ , ... ]  
... FROM { username | role }[ , ... ]  
... [ CASCADE ]
```

Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
CREATE	Revokes the right to create schemas in the specified database.
TEMP	Revokes the right to create temp tables in the database.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
<i>database-name</i>	Identifies the database from which to revoke the privilege.
<i>username</i>	Identifies the user from whom to revoke the privilege.
<i>role</i>	Identifies the role from which to revoke the privilege.
CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples

The following example revokes Fred's right to create schemas on vmartdb:

```
=> REVOKE CREATE ON DATABASE vmartdb FROM Fred;
```

The following revokes Fred's right to create temporary tables in vmartdb:

```
=> REVOKE TEMP ON DATABASE vmartdb FROM Fred;
```

See Also

- [GRANT \(Database\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Library)

Revokes the USAGE privilege on a library from a user or role.

To revoke functions inside the library, a user must have separate REVOKE privileges for those functions.

Syntax

```
REVOKE { USAGE | ALL }  
... ON LIBRARY [ [ db-name.]schema.]library-name [ , ... ]  
... FROM { username | PUBLIC | role } [ , ... ]  
...[ CASCADE ]
```

Parameters

<code>[[<i>db-name</i>.]<i>schema</i>.]</code>	Specifies a schema and an optional database to which you are connected. Include the schema name if multiple schemas exist in the database.
<code><i>library-name</i></code>	The library from which to revoke the USAGE privilege. When using more than one schema, specify the schema that contains the procedure.

<code>username role PUBLIC } [,...]</code>	<p>The area from which the privilege gets revoked, valid values are:</p> <ul style="list-style-type: none">• <code>username</code>—revokes privileges from a specific user• <code>role</code>—revokes privileges from a role• <code>PUBLIC</code>—revokes privileges from all users and roles.
<code>CASCADE</code>	<p>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The <code>CASCADE</code> keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</p>

Privileges

You must have `DBADMIN` privileges to run `REVOKE` (Library).

Examples

This example revokes `user1`'s `USAGE` privilege on the `idolLib` library in the `v_idol` schema.

```
=> REVOKE USAGE ON LIBRARY v_idol.IdolLib FROM user1;
```

See Also

- [GRANT \(Library\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Model)

Revokes privileges on a model from a user or role.

Syntax

```
REVOKE { ALL | USAGE }  
... ON MODEL [ [ db-name.]schema.]modelName [ , ... ]  
... FROM { username | role | PUBLIC} [ , ... ]  
... [ CASCADE ]
```

Parameters

<code>[[<i>db-name</i>.]<i>schema</i>]</code>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema</pre>
<code><i>modelName</i></code>	<p>Specifies the model from which to remove privileges.</p>
<code><i>username</i>] <i>role</i>] PUBLIC</code>	<p>The name of the user, role or group to be granted access.</p>
<code>CASCADE</code>	<p>Revokes the privilege from the</p>

specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Example

This example revokes user1's USAGE privilege on the mySvmClassModel model:

```
=> REVOKE USAGE ON mySvmClassModel FROM user1;
```

See Also

- [GRANT \(Model\)](#)
- [Managing Model Security](#)

REVOKE (Procedure)

Revokes the execute privilege on a procedure from a user or role.

Syntax

```
REVOKE EXECUTE
... ON PROCEDURE [ schema.]procedure-name[,...]
... ( [ argname ] argtype[,... ] )
... FROM { PUBLIC | { username | role }[,... ] }
...[ CASCADE ]
```

Parameters

<i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>procedure-name</i>	Specifies the procedure on which to revoke the execute privilege.
<i>argname</i>	Optionally specifies the argument names used when creating the procedure.
<i>argtype</i>	Specifies the argument types used when creating the procedure.
PUBLIC	Revokes the privilege from all users.
<i>username</i>	Specifies the user from whom to revoke the privilege.
<i>role</i>	Specifies the role from whom to revoke the privilege.
CASCADE	Revokes the privilege from the specified user or role and then from

others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Privileges

You must have DBADMIN privileges to run REVOKE (Procedure).

Examples

This example revokes Bob's execute privilege on the tokenize procedure.

```
=> REVOKE EXECUTE ON PROCEDURE tokenize(varchar) FROM Bob;
```

See Also

- [GRANT \(Procedure\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Resource Pool)

Revokes a user's or role's access privilege to a resource pool.

Syntax

```
REVOKE USAGE... ON RESOURCE POOL resource-pool  
... FROM { username | PUBLIC | role } [ , ... ]  
...[ CASCADE ]
```

Parameters

<i>resource-pool</i>	Specifies the resource pool from which to revoke the usage privilege.
<i>username</i>	Revokes the privilege from the specified user.

PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privilege from the specified role.
CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Notes

- Vertica checks privileges on resource pools during runtime. If the user running a query does not have the USAGE privilege on the appropriate pool, the query fails with an error. In this case, the user can use ALTER USER ... RESOURCE POOL to access another resource pool. Note the user may need to be granted usage on the resource pool with GRANT USAGE ON RESOURCE POOL.
- Revoking a user's permission from a resource pool in another session affects the user's access current session. In this case, the current session fails with an error stating that the resource pool in the current session does not exist.

Examples

This example shows how to revoke user Joe's usage privilege on the Joe_pool resource pool.

```
=> REVOKE USAGE ON RESOURCE POOL Joe_pool FROM Joe;  
REVOKE PRIVILEGE
```

See Also

- [GRANT \(Resource Pool\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Role)

Revokes a role (and administrative access, applicable) from a grantee. A user that has administrator access to a role can revoke the role for other users.

You can also remove a role's access to another role.

Syntax

```
REVOKE [ ADMIN OPTION FOR ] role [, ...]  
... FROM { user | role | PUBLIC } [, ...]  
...[ CASCADE ];
```

Parameters

ADMIN OPTION FOR	Revokes just the user's or role's administration access to the role, and not the role itself.
<i>role</i>	The name of one or more roles from which you want to revoke access.
<i>user</i> <i>role</i> PUBLIC	The name of a user or role whose permission you want to revoke. You can use the PUBLIC option to revoke access to a role that was previously made public.
CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples

This example shows the revocation of the pseudosuperuser role from the dbadmin user:

```
=> REVOKE pseudosuperuser from dbadmin;
```

This example shows the revocation of administration access from the dbadmin user for the pseudosuperuser role. The ADMIN OPTION command does not remove the pseudosuperuser role.

```
=> REVOKE ADMIN OPTION FOR pseudosuperuser FROM dbadmin;
```

Notes

If the role you are trying to revoke was not already granted to the user, Vertica returns a NOTICE:

```
=> REVOKE commentor FROM Sue;
NOTICE 2022: Role "commentor" was not already granted to user "Sue"
REVOKE ROLE
```

See Also

- [GRANT \(Role\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Schema)

Revokes privileges on a schema from a user or role.

Note: In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

Syntax

```
REVOKE [ GRANT OPTION FOR ] {
... { CREATE | USAGE } [ ,... ]
.. { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [ ,... ] ... | ALL [ PRIVILEGES ] }
... ON SCHEMA [db-name.] schema [ , ... ]
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
------------------	--

CREATE	Revokes the right to create tables and views in the schema.
USAGE	Revokes user access to the objects contained in the schema. Note that the user can also have access to the individual objects revoked. See the GRANT TABLE and GRANT VIEW statements.
SELECT	Revokes the user's ability to perform a SELECT on any column of any table in the schema.
INSERT	Revokes the user's ability to INSERT tuples into tables in the schema and use the COPY command to load data into the tables.
UPDATE	Revokes the user's ability to UPDATE tuples in a schema table.
DELETE	Revokes the user's ability to DELETE rows from a schema table.
REFERENCES	Revokes the user's privilege on both the referencing and referenced tables for creating a foreign key constraint.
ALL	Revokes all privileges previously granted.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[<i>db-name.</i>]	[Optional] Specifies the current database name. Using a database name prefix is optional, and does not affect the command in any way. You must be connected to the specified database.
<i>schema</i>	Identifies the schema from which to revoke privileges.
<i>username</i>	Revokes the privilege to a specific user.
PUBLIC	Revokes the privilege to all users.
<i>role</i>	Revokes the privilege to a specific role.
CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples

This example shows how to revoke user Joe's usage privileges on the online_sales schema.

```
=> REVOKE USAGE ON SCHEMA online_sales FROM Joe;
REVOKE PRIVILEGE
```

See Also

- [GRANT \(Schema\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Sequence)

Revokes privileges on a sequence generator from a user or role. Optionally revokes privileges on all sequences within one or more schemas.

Syntax

```
REVOKE [ GRANT OPTION FOR ]
... { SELECT | ALL [ PRIVILEGES ] }
... ON SEQUENCE [ [ db-name.]schema.]sequence-name [ , ... ]
... | ON ALL SEQUENCES IN SCHEMA schema-name [ , ... ]
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

Parameters

SELECT	Revokes the right to use both the CURRVAL and NEXTVAL functions on the specified sequence.
ALL	Applies to all privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[<i>db-name</i> .] <i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>
<i>sequence-name</i>	Specifies the sequence from which to revoke privileges.
ON ALL SEQUENCES IN SCHEMA	Revokes privileges on all sequences within one or more schemas from a user and/or role.

<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privilege from the specified role.
CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples

This example shows how to revoke user Joe's privileges on the my_seq sequence.

```
=> REVOKE ALL PRIVILEGES ON SEQUENCE my_seq FROM Joe;  
REVOKE PRIVILEGE
```

See Also

- [GRANT \(Sequence\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (Storage Location)

Revokes privileges from a user or role to read from or write to a storage location.

Note: The REVOKE (Storage Location) statement is applicable only to 'USER' storage locations. See [GRANT \(Storage Location\)](#) for more information. If the storage location is dropped, all user privileges are removed as part of that.

Syntax

```
REVOKE [ GRANT OPTION FOR ]  
... { READ | WRITE | ALL [ PRIVILEGES ] }  
... ON LOCATION [ 'path' [ ON node ]  
... FROM { username | role | PUBLIC } [, ...]  
... [ CASCADE ]
```

Parameters

REVOKE OPTION FOR	Revokes GRANT privileges from the grantee.
READ	Revokes privileges from the grantee to copy data from files in a storage locations into a table.
WRITE	Revokes privileges to export Vertica data from the database to a storage location. Revoking WRITE privileges prevents the grantee from exporting COPY statement rejected data and exceptions files to a storage location.
ALL	Revokes all privileges from the grantee to the storage location.
PRIVILEGES	For SQL standard compatibility and is ignored.
{ <i>username</i> <i>role</i> PUBLIC } [,...]	<p>Revokes privileges from one or more users, one or more roles, or all users (PUBLIC).</p> <ul style="list-style-type: none"> • <i>username</i> – A specific user • <i>role</i> – A particular role • PUBLIC – Revokes privileges from all users and roles.
ON LOCATION ('path' [ON node])	<ul style="list-style-type: none"> • <i>path</i>— Specifies the path name mount point of the storage location • <i>node</i>— [Optional] Revokes access to the storage location residing on the node. If you leave this blank, <i>node</i> defaults to all nodes on the specified path in that cluster. If a path exists for only some nodes, the entire grant rolls back, even on the nodes that reside in the path.
CASCADE	Revokes privileges from the grantee. Using the CASCADE keyword when revoking privileges first removes them from the initial grantee. The statement then revokes all privileges from other users and roles to whom the grantee extended storage location access.

Examples

For examples, see [GRANT \(Storage Location\)](#)

See Also

- [Granting and Revoking Privileges](#)

REVOKE (Table)

Revokes privileges on a table from a user or role. Optionally revokes privileges on all tables within one or more schemas.

Note: Revoking privileges on all tables within a schema includes all views in the same schema.

In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

Syntax

```
REVOKE [ GRANT OPTION FOR ]... {  
.....{ SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [ ,... ]  
.....| ALL [ PRIVILEGES ]  
... }  
... ON [ TABLE ] [ [ db-name.]schema.]tablename [ , ... ]  
... | ON ALL TABLES IN SCHEMA schema-name [ , ... ]  
... FROM { username | PUBLIC | role } [ , ... ]  
... [ CASCADE ]
```

Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
SELECT	Revokes the user's ability to SELECT from any column of the specified table.
INSERT	Revokes the user from being able to INSERT tuples into the

	<p>specified table and to use the COPY command to load the table.</p> <p>Note: COPY FROM STDIN is allowed to any user granted the INSERT privilege, while COPY FROM <file> is an admin-only operation.</p>
UPDATE	Revokes user from being allowed to UPDATE tuples in the specified table.
DELETE	Revokes user from being able to DELETE a row from the specified table.
REFERENCES	Revokes the user's privilege on both the referencing and referenced tables for creating a foreign key constraint.
TRUNCATE	<p>Revokes TRUNCATE TABLE privileges from non-owners of a table. It also revokes from the non-owner of the table the following partition functions:</p> <ul style="list-style-type: none"> • DROP_PARTITION • SWAP_PARTITIONS_BETWEEN_TABLES • MOVE_PARTITIONS_TO_TABLE
ALL	Revokes all previously granted privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[<i>db-name.</i>] <i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>tableName</i>	Specifies the table from which to remove privileges.
ON ALL TABLES IN SCHEMA	Revokes privileges on all tables (and by default views) within one or more schemas from a user and/or role.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
<i>role</i>	Revokes the privilege from the specified role.
CASCADE	Revokes the privilege from the specified user or role and then from

others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples

This example shows how to revoke user Joe's privileges on the customer_dimension table.

```
=> REVOKE ALL PRIVILEGES ON TABLE customer_dimension FROM Joe;  
REVOKE PRIVILEGE
```

See Also

- [GRANT \(Table\)](#)
- [Granting and Revoking Privileges](#)

REVOKE (User Defined Extension)

Revokes the EXECUTE privilege on a user-defined extension (UDx) from a database user or role. Optionally revokes privileges on all user-defined extensions within one or more schemas. You can revoke privileges on the following user-defined extension types:

- User Defined Functions (UDF)
 - User Defined SQL Functions
 - User Defined Scalar Functions (UDSF)
 - User Defined Transform Functions (UDTF)
 - User Defined Aggregate Functions (UDAF)
 - User Defined Analytic Functions (UDAnF)
- User Defined Load Functions (UDL)
 - UDL Filter

- UDL Parser
- UDL Source

Syntax

```

REVOKE EXECUTE
...ON FUNCTION [ schema.]function-name [, ...]
... | ON AGGREGATE FUNCTION [ schema.]function-name [, ...]
... | ON ANALYTIC FUNCTION [ schema.]function-name [, ...]
... | ON TRANSFORM FUNCTION [ schema.]function-name [, ...]
... | ON FILTER [ schema.]filter-name [, ...]
... | ON PARSER [ schema.]parser-name [, ...]
... | ON SOURCE [ schema.]source-name [, ...]
... | ON ALL FUNCTIONS IN SCHEMA schema-name [, ...]
... ( [ argname ] argtype [, ...] )
... FROM { username | role | PUBLIC } [, ...]
... [ CASCADE ]

```

Parameters

<i>schema</i>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<i>function-name</i> <i>filter-name</i> <i>parser-name</i> <i>source-name</i>	The name of the UDX from which to revoke the EXECUTE privilege. If you use more than one schema, you must specify the schema that contains the UDX, as noted in the previous row.
ON ALL FUNCTIONS IN SCHEMA	Revokes EXECUTE privileges on all UDX's within one or more schemas from a user, role, or all users and roles.
<i>argname</i>	[Optional] The argument name or names for the UDX. When you GRANT , REVOKE , or DROP privileges for a polymorphic function, you must include an argument with the command.
<i>argtype</i>	The argument data type or types of the UDX.
{ <i>username</i> <i>role</i> PUBLIC } [, ...]	Revokes the privileges from the specified user, role, or all users and roles (PUBLIC) that have been granted the privilege.

CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.
---------	--

Privileges

You must have DBADMIN privileges, or be the owner of the UDX, to run REVOKE (User Defined Extension).

Examples

The following command revokes EXECUTE privileges from user Bob on the myzeroifnull function:

```
> REVOKE EXECUTE ON FUNCTION myzeroifnull (x INT) FROM Bob;
```

The following command revokes ALL privileges from user Doug on the Pagerank polymorphic function:

```
> REVOKE ALL ON TRANSFORM FUNCTION Pagerank (t float) FROM Doug;
```

The following command revokes EXECUTE privileges on all functions in the zero-schema schema from user Bob:

```
> REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA zero-schema FROM Bob;
```

The following command revokes EXECUTE privileges from user Bob on the tokenize function:

```
> REVOKE EXECUTE ON TRANSFORM FUNCTION tokenize(VARCHAR) FROM Bob;
```

The following command revokes ALL privileges on the ExampleSource() source from user Alice:

```
> REVOKE ALL ON SOURCE ExampleSource() FROM Alice;
```

See Also

- [GRANT \(User Defined Extension\)](#)
- [Granting and Revoking Privileges](#) in the Administrator's Guide
- [Developing User-Defined Extensions \(UDxs\)](#) in Extending Vertica

REVOKE (View)

Revokes user privileges on a view.

Important: In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

Syntax

```
REVOKE [ GRANT OPTION FOR ]...  
    { SELECT | ALL [ PRIVILEGES ] }  
... ON [ [ db-name. ] schema. ] viewname [ , ... ]  
... FROM { username | PUBLIC } [ , ... ]  
... [ CASCADE ]
```

Parameters

GRANT OPTION FOR	Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.
SELECT	Revokes the user's ability to perform SELECT operations on a view and its referenced resources.
ALL	Revokes all previously granted privileges.
PRIVILEGES	Is for SQL standard compatibility and is ignored.
[<i>db-name</i> .] <i>schema</i>	Specifies a schema . If multiple schemas are defined in the database, include the schema name. For example: <pre>myschema.thisDBObject</pre>

<i>viewname</i>	Specifies the view on which to revoke the privileges.
<i>username</i>	Revokes the privilege from the specified user.
PUBLIC	Revokes the privilege from all users.
CASCADE	Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples

This example shows how to revoke SELECT privileges from user Joe on the view named test_view.

```
=> REVOKE SELECT ON test_view FROM Joe;  
REVOKE PRIVILEGE
```

You can also use the database name and schema name in the command:

```
=> REVOKE SELECT ON VMart.public.test_view FROM Joe;  
REVOKE PRIVILEGE
```

See Also

- [GRANT \(View\)](#)
- [Granting and Revoking Privileges](#)

ROLLBACK

Ends the current transaction and discards all changes that occurred during the transaction.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ]
```

Parameters

WORK | TRANSACTION

Have no effect; they are optional keywords for readability.

Privileges

No special permissions required.

Notes

When an operation is rolled back, any locks that are acquired by the operation are also rolled back.

ABORT is a synonym for ROLLBACK.

Examples

This example shows how to roll back from a DELETE transaction.

```
=> SELECT * FROM sample_table;
a
---
1
(1 row)

=> DELETE FROM sample_table WHERE a = 1;

=> SELECT * FROM sample_table;
a
---
(0 rows)

=> ROLLBACK;

=> SELECT * FROM sample_table;
a
---
1
(1 row)
```

This example shows how to roll back the changes you made since the BEGIN statement.

```
=> BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;  
BEGIN  
  
=> ROLLBACK TRANSACTION;  
ROLLBACK
```

See Also

- [Transactions](#)
- [Creating Transactions](#)
- [BEGIN](#)
- [COMMIT](#)
- [END](#)
- [START TRANSACTION](#)

ROLLBACK TO SAVEPOINT

Rolls back all commands that have been entered within the transaction since the given savepoint was established.

Syntax

```
ROLLBACK TO [SAVEPOINT] savepoint_name
```

Parameters

<i>savepoint_name</i>	Specifies the name of the savepoint to roll back to.
-----------------------	--

Privileges

No special permissions required.

Notes

- The savepoint remains valid and can be rolled back to again later if needed.
- When an operation is rolled back, any locks that are acquired by the operation are also rolled back.
- ROLLBACK TO SAVEPOINT implicitly destroys all savepoints that were established after the named savepoint.

Example

The following example rolls back the values 102 and 103 that were entered after the savepoint, `my_savepoint`, was established. Only the values 101 and 104 are inserted at commit.

```
=> INSERT INTO product_key VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (102);
=> INSERT INTO product_key VALUES (103);
=> ROLLBACK TO SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (104);
=> COMMIT;
```

See Also

- [RELEASE SAVEPOINT](#)
- [SAVEPOINT](#)

SAVEPOINT

Creates a special mark, called a savepoint, inside a transaction. A savepoint allows all commands that are executed after it was established to be rolled back, restoring the transaction to the state it was in at the point in which the savepoint was established.

Tip: Savepoints are useful when creating nested transactions. For example, a savepoint could be created at the beginning of a subroutine. That way, the result of the subroutine could be rolled back if necessary.

Syntax

`SAVEPOINT savepoint_name`

Parameters

<code><i>savepoint_name</i></code>	Specifies the name of the savepoint to create.
------------------------------------	--

Privileges

No special permissions required.

Notes

- Savepoints are local to a transaction and can only be established when inside a transaction block.
- Multiple savepoints can be defined within a transaction.
- If a savepoint with the same name already exists, it is replaced with the new savepoint.

Example

The following example illustrates how a savepoint determines which values within a transaction can be rolled back. The values 102 and 103 that were entered after the savepoint, `my_savepoint`, was established are rolled back. Only the values 101 and 104 are inserted at commit.

```
=> INSERT INTO T1 (product_key) VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO T1 (product_key) VALUES (102);
=> INSERT INTO T1 (product_key) VALUES (103);
=> ROLLBACK TO SAVEPOINT my_savepoint;
=> INSERT INTO T1 (product_key) VALUES (104);
=> COMMIT;
=> SELECT product_key FROM T1;
.
.
```

```
.  
101  
104  
(2 rows)
```

See Also

- [RELEASE SAVEPOINT](#)
- [ROLLBACK TO SAVEPOINT](#)

SELECT

Retrieves a result set from one or more tables.

Syntax

```
[{ AT EPOCH {LATEST | epoch-number} | AT TIME 'timestamp'}]  
SELECT [ /*+ LABEL(Label-name)* / ] [ ALL | DISTINCT ]  
... { * | expression [ [AS] output-name ] },... ]  
... [ INTO TABLE ]  
... [ from-clause ]  
... [ WHERE condition ]  
... [ TIMESERIES slice-time ]  
... [ GROUP BY expression[,...] ]  
... [ HAVING condition[,...] ]  
... [ MATCH ]  
... [ UNION { ALL | DISTINCT } ]  
... [ EXCEPT ]  
... [ INTERSECT ]  
... [ ORDER BY expression { ASC | DESC }[,,...] ]  
... [ LIMIT { count | ALL } ]  
... [ OFFSET start ]  
... [ FOR UPDATE [ OF table-name[,...] ] ]
```

Parameters

Note: SELECT clauses such as INTO and WHERE are discussed in sub-sections of this page.

AT EPOCH LATEST	Queries all data in the database up to but not including the current
-----------------	--

	<p>epoch without holding a lock or blocking write operations. See Snapshot Isolation for more information. AT EPOCH LATEST is ignored when applied to temporary tables (all rows are returned).</p> <p>By default, queries run under the READ COMMITTED isolation level, which means:</p> <ul style="list-style-type: none"> • AT EPOCH LATEST includes data from the latest committed DML transaction. • Each epoch contains exactly one transaction—the one that modified the data. • The Tuple Mover can perform moveout and mergeout operations on committed data immediately.
<i>epoch-number</i>	Queries all data in the database up to and including the current epoch without holding a lock or blocking write operations.
AT TIME ' <i>timestamp</i> '	<p>Specifies an historical query, where all data in the database is queried, up to and including the epoch representing the specified date and time. This query holds no locks and blocks no write operations.</p> <p>AT TIME is ignored when applied to temporary tables (all rows are returned).</p>
<i>/*+LABEL (Label-name)*</i>	<p>Assigns a label to a query so you can identify it for profiling and debugging.</p> <p>In a UNION statement, only the first SELECT statement can be labeled; Vertica ignores labels in subsequent SELECT statements.</p>
ALL DISTINCT	<ul style="list-style-type: none"> • ALL (default): Retains duplicate rows in result set or group. • DISTINCT: Removes duplicate rows from the result set or group. <p>The ALL or DISTINCT qualifier must immediately follow the SELECT keyword. Only one instance of this keyword can appear in the select list.</p>
*	<p>Lists all columns in the queried tables.</p> <p>Caution: Selecting all columns from the queried tables can</p>

	produce a very large wide set, which can adversely affect performance.
<i>expression</i> [[AS] <i>output-name</i>]	A table column or column expression to select from the queried tables. You can optionally qualify <i>expression</i> with an output name, which can be used in several ways: <ul style="list-style-type: none">• Label the column for display.• Refer to the column's value in ORDER BY and GROUP BY clauses (it cannot be referenced in WHERE or HAVING clauses).
<i>from-clause</i>	A comma-separated list of data sources to query.
FOR UPDATE	Specifies to obtain an X lock on all tables specified in the query, most often used from READ COMMITTED isolation. FOR UPDATE requires update/delete permissions on the queried tables and cannot be issued from a read-only transaction.

Privileges

You must have USAGE privileges on the schemas that contain the queried tables, as well as one, but not both, of the following:

- Owner or user with GRANT OPTION privileges
- SELECT privilege

If the SELECT statement queries a view, the view owner must have SELECT privileges on the view's anchor tables or views.

Example

When multiple clients run transactions as in the following example query, deadlocks can occur if FOR UPDATE is not used. Two transactions acquire an S lock, and when both attempt to upgrade to an X lock, they encounter deadlocks:

```
=> SELECT balance FROM accounts WHERE account_id=3476 FOR UPDATE;  
...  
=> UPDATE accounts SET balance = balance+10 WHERE account_id=3476;
```

```
=> COMMIT;
```

See Also

- [LOCKS](#)
- [Analytic Functions](#)
- [SQL Analytics](#)
- [Time Series Analytics](#)
- [Event Series Pattern Matching](#)
- [Subqueries](#)
- [Joins](#)

EXCEPT Clause

Combines two or more SELECT queries. EXCEPT returns distinct results of the left-hand query that are not also found in the right-hand query.

Note: MINUS is an alias for EXCEPT.

Syntax

```
SELECT  
... EXCEPT select  
... [ EXCEPT select ]...  
... [ ORDER BY { column-name  
... | ordinal-number }  
... [ ASC | DESC ] [ , ... ] ]  
... [ LIMIT { integer | ALL } ]  
... [ OFFSET integer ]
```

Notes

- Use the EXCEPT clause to filter out specific results from a SELECT statement. The EXCEPT query operates on the results of two or more SELECT queries. It returns only those rows in

the left-hand query that are not also present in the right-hand query.

- Vertica evaluates multiple EXCEPT clauses in the same SELECT query from left to right, unless parentheses indicate otherwise.
- You cannot use the ALL keyword with an EXCEPT query.
- The results of each SELECT statement must be union compatible. Each statement must return the same number of columns, and the corresponding columns must have compatible data types. For example, you cannot use the EXCEPT clause on a column of type INTEGER and a column of type VARCHAR. If statements do not meet these criteria, Vertica returns an error.

Note: The [Data Type Coercion Chart](#) lists the data types that can be cast to other data types. If one data type can be cast to the other, those two data types are compatible.

- You can use EXCEPT in FROM, WHERE, and HAVING clauses.
- You can order the results of an EXCEPT operation by including an ORDER BY operation in the statement. When you write the ORDER BY list, specify the column names from the leftmost SELECT statement, or specify integers that indicate the position of the columns by which to sort.
- The rightmost ORDER BY, LIMIT, or OFFSET clauses in an EXCEPT query do not need to be enclosed in parentheses, because the rightmost query specifies that Vertica perform the operation on the results of the EXCEPT operation. Any ORDER BY, LIMIT, or OFFSET clauses contained in SELECT queries that appear earlier in the EXCEPT query must be enclosed in parentheses.
- Vertica supports EXCEPT noncorrelated subquery predicates. For example:

```
=> SELECT * FROM T1
    WHERE T1.x IN
      (SELECT MAX(c1) FROM T2
       EXCEPT
       SELECT MAX(cc1) FROM T3
       EXCEPT
       SELECT MAX(d1) FROM T4);
```

Examples

Consider the following three tables:

Company_A

Id	emp_lname	dept	sales
1234	Stephen	auto parts	1000
5678	Alice	auto parts	2500
9012	Katherine	floral	500
3214	Smithson	sporting goods	1500

(4 rows)

Company_B

Id	emp_lname	dept	sales
4321	Marvin	home goods	250
8765	Bob	electronics	20000
9012	Katherine	home goods	500
3214	Smithson	home goods	1500

(4 rows)

Company_C

Id	emp_lname	dept	sales
3214	Smithson	sporting goods	1500
5432	Madison	sporting goods	400
7865	Cleveland	outdoor	1500
1234	Stephen	floral	1000

(4 rows)

The following query returns the IDs and last names of employees that exist in Company_A, but not in Company_B:

```
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT id, emp_lname FROM Company_B;
id | emp_lname
-----+-----
1234 | Stephen
5678 | Alice
(2 rows)
```

The following query sorts the results of the previous query by employee last name:

```
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT id, emp_lname FROM Company_B
      ORDER BY emp_lname ASC;
id | emp_lname
-----+-----
5678 | Alice
1234 | Stephen
(2 rows)
```

If you order by the column position, the query returns the same results:

```
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT id, emp_lname FROM Company_B
      ORDER BY 2 ASC;
 id | emp_lname
-----+-----
5678 | Alice
1234 | Stephen
(2 rows)
```

The following query returns the IDs and last names of employees that exist in Company_A, but not in Company_B or Company_C:

```
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT id, emp_lname FROM Company_B
      EXCEPT
      SELECT id, emp_lname FROM Company_C;
 id | emp_lname
-----+-----
5678 | Alice
(1 row)
```

The following query shows the results of mismatched data types:

```
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT emp_lname, id FROM Company_B;
ERROR 3429: For 'EXCEPT', types int and varchar are inconsistent
DETAIL: Columns: id and emp_lname
```

Using the [VMart](#) example database, the following query returns information about all Connecticut-based customers who bought items through stores and whose purchases amounted to more than \$500, except for those customers who paid cash:

```
=> SELECT customer_key, customer_name FROM public.customer_dimension
      WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact
                            WHERE sales_dollar_amount > 500
                            EXCEPT
                            SELECT customer_key FROM store.store_sales_fact
                            WHERE tender_type = 'Cash')
      AND customer_state = 'CT';
 customer_key | customer_name
-----+-----
15084 | Doug V. Lampert
21730 | Juanita F. Peterson
24412 | Mary U. Garnett
25840 | Ben Z. Taylor
29940 | Brian B. Dobisz
32225 | Ruth T. McNulty
33127 | Darlene Y. Rodriguez
40000 | Steve L. Lewis
44383 | Amy G. Jones
46495 | Kevin H. Taylor
(10 rows)
```

See Also

- [SELECT](#)
- [INTERSECT Clause](#)
- [UNION Clause](#)
- [Subqueries](#)

FROM Clause

A comma-separated list of data sources to query.

Syntax

```
FROM dataset [,... ] [ TABLESAMPLE(sampling_percent) ]
```

Parameters

<i>dataset</i>	<p>A set of data to query, which can be one of the following elements:</p> <ul style="list-style-type: none">• Table reference• Joined tables• Named subquery <p><i>subquery</i> [AS] <i>name</i></p>
TABLESAMPLE	<p>A clause after <i>dataset</i> indicating that simple random sampling should be used to return an approximate percentage of records. Each row of the data set has the same opportunity to be selected. Vertica performs this sampling before other filters in the query are applied. The number of records returned is not guaranteed to be the exact percentage of records defined by <i>sampling_percent</i>.</p> <p>You can only use the TABLESAMPLE clause with user-defined tables. Views, Data Collector (DC), and system tables are not supported.</p>

<i>sampling_percent</i>	Specifies the percentage of records to be returned as a part of the sampling. The value must be greater than 0 and less than 100.
-------------------------	---

Examples

The following example shows how you can return all records from the `customer_dimension` table:

```
=> SELECT * FROM customer_dimension;
```

The following example shows how you can use the `TABLESAMPLE` clause:

```
=> SELECT * FROM user_name TABLESAMPLE(50);
 id | name
-----+-----
 111 | Barry
  982 | Nero
 8761 | Lou
(3 rows)
```

Table-Reference

Syntax

table-name [[AS] *alias*]

Parameters

<i>table-name</i>	A table in the logical schema.
[AS] <i>alias</i>	A temporary name used for references to <i>table-name</i> .

Joined-Table

Specifies how to join tables.

Syntax

table-reference [join-type] JOIN *table-reference*[TABLESAMPLE(*sampling-pct*)] [ON *join-predicate*]

Parameters

<i>table-reference</i>	A table or another <i>joined-table</i> .
<i>join-type</i>	<p>Valid Values:</p> <ul style="list-style-type: none"> • INNER (default). INNER JOIN is equivalent to a query that specifies its join predicate in a WHERE clause. • LEFT [OUTER] • RIGHT [OUTER] • FULL [OUTER] • NATURAL • CROSS
TABLESAMPLE	<p>Specifies to use simple random sampling to return an approximate percentage of records. All rows in the total potential return set are equally eligible to be included in the sampling. Vertica performs this sampling before other filters in the query are applied. The number of records returned is not guaranteed to be the exact percentage of records defined by <i>sampling-pct</i>.</p> <p>The TABLESAMPLE option is valid only with user-defined tables and Data Collector (DC) tables. Views and system tables are not supported.</p>
<i>sampling-pct</i>	Specifies the percentage of records to be returned as a part of sampling. The value must be greater than 0 and less than 100.
ON <i>join-predicate</i>	An equi-join based on one or more columns in the joined tables. invalid for NATURAL and CROSS joins, required for all other join types.

Alternative JOIN Syntax Options

Vertica supports two older join syntax conventions:

- Table joins specified by join predicate in a WHERE clause
- Table joins specified by a USING clause

For details, see [Join Syntax](#) in Analyzing Data.

Examples

The following SELECT statement qualifies its JOIN clause with the TABLESAMPLE option:

```
=> SELECT user_id.id, user_name.name FROM user_name TABLESAMPLE(50)
      JOIN user_id TABLESAMPLE(50) ON user_name.id = user_id.id;
 id | name
-----+-----
 489 | Markus
 2234 | Cato
  763 | Pompey
(3 rows)
```

GROUP BY Clause

Use the GROUP BY clause with aggregate functions in a SELECT statement to collect data across multiple records. Vertica groups the results into one or more sets of rows that match an expression.

The GROUP BY clause without aggregates is similar to using SELECT DISTINCT.

[ROLLUP](#) is an extension to the GROUP BY clause. ROLLUP performs subtotal aggregations.

Syntax

```
GROUP BY [/*+GBYTYPE(algorithm)*/] expression-or-aggregate-expression [ ,... ]
```

Arguments

```
/*+GBYTYPE  
(  
algorithm)*/
```

Specifies which algorithm has precedence for implementing this GROUP BY clause, over the algorithm the Vertica query optimizer might otherwise choose. You can set *algorithm* to one of the following values:

- HASH: GROUPBY HASH algorithm

	<ul style="list-style-type: none"> • PIPE: GROUPBY PIPELINED algorithm <p>For more information about both algorithms, see GROUP BY Implementation Options.</p>
<p><i>expression-or-aggregate-expression</i></p>	<p>An <i>expression</i> is:</p> <ul style="list-style-type: none"> • Any expression, including constants and column references in the tables specified in the FROM clause. For example: <pre>column1, ..., column_n, (expression)</pre> <p>An <i>aggregate-expression</i> is:</p> <ul style="list-style-type: none"> • An ordered list of columns, expressions, CUBE , GROUPING SETS, or ROLLUP aggregates. <p>You can include CUBE and ROLLUP aggregates within a GROUPING SETS aggregate. CUBE and ROLLUP aggregates can result in a large amount of output. In that case, use GROUPING SETS to return only certain results.</p> <p>You cannot include any aggregates within a CUBE or ROLLUP expression.</p> <p>You can append multiple GROUPING SETS, CUBE, or ROLLUP aggregates in the same query. Examples:</p> <pre>GROUP BY a,b,c,d, ROLLUP(a,b) GROUP BY a,b,c,d, CUBE((a,b),c,d) GROUP BY a,b,c,d, CUBE(a,b), ROLLUP (c,d) GROUP BY ROLLUP(a), CUBE(b), GROUPING SETS(c) GROUP BY a,b,c,d, GROUPING SETS ((a,d),(b,c),CUBE(a,b)) GROUP BY a,b,c,d, GROUPING SETS ((a,d),(b,c),(a,b),(a),(b),())</pre>

Usage Considerations

- *expression* cannot include [aggregate functions](#). However, you can use the GROUP BY clause with CUBE, GROUPING SETS, and ROLLUP to return summary values for each group.
- When you create a GROUP BY clause, you must include all non-aggregated columns that appear in the SELECT list.

- If the GROUP BY clause includes a WHERE clause, Vertica ignores all rows that do not satisfy the WHERE clause.

Examples

This example shows how to use the WHERE clause with GROUP BY. In this case, the example retrieves all employees whose last name begins with S, and ignores all rows that do not meet this criteria. The GROUP BY clause uses the ILIKE function to retrieve only last names beginning with S. The aggregate function SUM computes the total vacation days for each group.

```
=> SELECT employee_last_name, SUM(vacation_days)
      FROM employee_dimension
      WHERE employee_last_name ILIKE 'S%'
      GROUP BY employee_last_name;
employee_last_name | SUM
-----+-----
Sanchez            | 2892
Smith              | 2672
Stein              | 2660
(3 rows)
```

The GROUP BY clause in the following example groups results by vendor region, and vendor region's biggest deal:

```
=> SELECT vendor_region, MAX(deal_size) AS "Biggest Deal"
      FROM vendor_dimension
      GROUP BY vendor_region;
vendor_region | Biggest Deal
-----+-----
East          | 990889
MidWest      | 699163
NorthWest    | 76101
South        | 854136
SouthWest    | 609807
West         | 964005
(6 rows)
```

The following query modifies the previous one with a HAVING clause, which specifies to return only groups whose maximum deal size exceeds \$900,000:

```
=> SELECT vendor_region, MAX(deal_size) as "Biggest Deal"
      FROM vendor_dimension
      GROUP BY vendor_region
      HAVING MAX(deal_size) > 900000;
vendor_region | Biggest Deal
-----+-----
East          | 990889
West         | 964005
(2 rows)
```

The GROUP BY clause without aggregates is similar to using SELECT DISTINCT. For example, the following two queries return the same results:

```
=> SELECT DISTINCT household_id FROM customer_dimension;  
=> SELECT household_id FROM customer_dimension GROUP BY household_id;
```

See Also

- [CUBE Aggregate](#)
- [GROUP_ID](#)
- [GROUPING](#)
- [GROUPING_ID](#)
- [GROUPING SETS Aggregate](#)
- [ROLLUP](#)

ROLLUP Aggregate

Automatically performs subtotal aggregations as an extension to the [GROUP BY](#) clause. ROLLUP performs these aggregations across multiple dimensions, at different levels, within a single SQL query.

You can use the ROLLUP clause with three grouping functions:

- [GROUPING](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)

Syntax

```
ROLLUP grouping-expression[, ...]
```

Parameters

<i>group-expression</i>	One or both of the following:
-------------------------	-------------------------------

	<ul style="list-style-type: none">• An expression that is not an aggregate or a grouping function that includes constants and column references in FROM-specified tables. For example: <code>column1, (column2+1), column3+column4</code>• A multilevel expression, one of the following:<ul style="list-style-type: none">■ ROLLUP■ CUBE■ GROUPING SETS
--	--

Restrictions

GROUP BY ROLLUP does not sort results. To sort data, an ORDER BY [clause](#) must follow the GROUP BY clause.

Levels of Aggregation

If n is the number of grouping columns, ROLLUP creates $n+1$ levels of subtotals and grand total. Because ROLLUP removes the right-most column at each step, specify column order carefully.

Suppose that ROLLUP(A, B, C) creates four groups:

- (A, B, C)
- (A, B)
- (A)
- ()

Because ROLLUP removes the right-most column at each step, there are no groups for (A, C) and (B, C).

If you enclose two or more columns in parentheses, GROUP BY treats them as a single entity. For example:

- ROLLUP(A, B, C) creates four groups:
 - (A, B, C)
 - (A, B)
 - (A)
 - ()

- ROLLUP((A, B), C) treats (A, B) as a single entity and creates three groups:

(A, B, C)
(A, B)
()

Example: Aggregating the Full Data Set

The following example shows how to use the GROUP BY clause to determine family expenses for electricity and books over several years. The SUM aggregate function computes the total amount of money spent in each category per year.

Suppose you have a table that contains information about family expenses for books and electricity:

```
=> SELECT * FROM expenses ORDER BY Category, Year;
Year | Category | Amount
-----+-----+-----
2005 | Books    | 39.98
2007 | Books    | 29.99
2008 | Books    | 29.99
2005 | Electricity | 109.99
2006 | Electricity | 109.99
2007 | Electricity | 229.98
```

For the expenses table, ROLLUP computes the subtotals in each category between 2005–2007:

- Books: \$99.96
- Electricity: \$449.96
- Grand total: \$549.92.

Use the ORDER BY clause to sort the results:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY ROLLUP(Category, Year) ORDER BY 1,2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Books    |      | 99.96
Electricity | 2005 | 109.99
Electricity | 2006 | 109.99
Electricity | 2007 | 229.98
Electricity |      | 449.96
          |      | 549.92
```

Example: Using ROLLUP with the HAVING Clause

This example shows how to use the [HAVING](#) clause with ROLLUP to restrict the GROUP BY results. The following query produces only those ROLLUP categories where year is subtotaled, based on the expression in the GROUPING function:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY ROLLUP(Category,Year) HAVING GROUPING(Year)=1
      ORDER BY 1, 2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    |     | 99.96
Electricity |     | 449.96
         |     | 549.92
```

The next example rolls up on (Category, Year), but not on the full results. The GROUPING_ID function specifies to aggregate less than three levels:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY ROLLUP(Category,Year) HAVING GROUPING_ID(Category,Year)<3
      ORDER BY 1, 2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Books    |     | 99.96
Electricity | 2005 | 109.99
Electricity | 2006 | 109.99
Electricity | 2007 | 229.98
Electricity |     | 449.96
```

See Also

- [Data Aggregation](#)
- [CUBE Aggregate](#)
- [GROUPING](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)
- [GROUP BY Clause](#)
- [GROUPING SETS Aggregate](#)

GROUP_ID

Uniquely identifies duplicate sets for GROUP BY queries that return duplicate grouping sets. This function returns one or more integers, starting with zero (0), as identifiers.

For the number of duplicates n for a particular grouping, GROUP_ID returns a range of sequential numbers, 0 to $n-1$. For the first each unique group it encounters, GROUP_ID returns the value 0. If GROUP_ID finds the same grouping again, the function returns 1, then returns 2 for the next found grouping, and so on.

Note: Use GROUP_ID only in SELECT statements that contain a **GROUP BY** aggregate: **CUBE**, **GROUPING SETS**, and **ROLLUP**.

Behavior Type

Immutable

Syntax

GROUP_ID ()

Examples

This example shows how GROUP_ID creates unique identifiers when a query produces duplicate groupings. For an expenses table, the following query groups the results by category of expense and year and rolls up the sum for those two columns. The results have duplicate groupings for category and NULL. The first grouping has a GROUP_ID of 0, and the second grouping has a GROUP_ID of 1.

```
=> SELECT Category, Year, SUM(Amount), GROUPING_ID(Category, Year),  
       GROUP_ID() FROM expenses GROUP BY Category, ROLLUP(Category,Year)  
       ORDER BY Category, Year, GROUPING_ID();
```

Category	Year	SUM	GROUPING_ID	GROUP_ID
Books	2005	39.98	0	0
Books	2007	29.99	0	0
Books	2008	29.99	0	0
Books		99.96	1	0
Books		99.96	1	1
Electricity	2005	109.99	0	0
Electricity	2006	109.99	0	0
Electricity	2007	229.98	0	0
Electricity		449.96	1	1
Electricity		449.96	1	0

See Also

- [CUBE Aggregate](#)
- [GROUPING](#)
- [GROUPING_ID](#)
- [GROUPING SETS Aggregate](#)
- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

GROUPING

Disambiguates the use of NULL values when GROUP BY queries with multilevel aggregates generate NULL values to identify subtotals in grouping columns. Such NULL values from the original data can also occur in rows. GROUPING returns 1, if the value of *expression* is:

- NULL, representing an aggregated value
- 0 for any other value, including NULL values in rows

Note: Use GROUPING only in SELECT statements that contain a [GROUP BY](#) aggregate: [CUBE](#), [GROUPING SETS](#), and [ROLLUP](#).

Behavior Type

Immutable

Syntax

GROUPING (*expression*)

Parameters

<i>expression</i>	An expression in the GROUP BY clause
-------------------	--------------------------------------

Examples

The following query uses the GROUPING function, taking one of the GROUP BY expressions as an argument. For each row, GROUPING returns one of the following:

- 0: The column is part of the group for that row
- 1: The column is not part of the group for that row

The 1 in the GROUPING(Year) column for electricity and books indicates that these values are subtotals. The right-most column values for both GROUPING(Category) and GROUPING(Year) are 1. This value indicates that neither column contributed to the GROUP BY. The final row represents the total sales.

```
=> SELECT Category, Year, SUM(Amount),
      GROUPING(Category), GROUPING(Year) FROM expenses
      GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
      Category | Year | SUM   | GROUPING | GROUPING
-----+-----+-----+-----+-----
Books        | 2005 | 39.98 | 0         | 0
Books        | 2007 | 29.99 | 0         | 0
Books        | 2008 | 29.99 | 0         | 0
Books        |      | 99.96 | 0         | 1
Electricity  | 2005 | 109.99 | 0         | 0
Electricity  | 2006 | 109.99 | 0         | 0
Electricity  | 2007 | 229.98 | 0         | 0
Electricity  |      | 449.96 | 0         | 1
              |      | 549.92 | 1         | 1
```

See Also

- [CUBE Aggregate](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)
- [GROUPING SETS Aggregate](#)
- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

GROUPING_ID

Concatenates the set of Boolean values generated by the [GROUPING](#) function into a bit vector. [GROUPING_ID](#) treats the bit vector as a binary number and returns it as a base-10 value that identifies the grouping set combination.

By using [GROUPING_ID](#) you avoid the need for multiple, individual [GROUPING](#) functions. [GROUPING_ID](#) simplifies row-filtering conditions, because rows of interest are identified using a single return from `GROUPING_ID = n`. Use [GROUPING_ID](#) to identify grouping combinations.

Note: Use [GROUPING_ID](#) only in `SELECT` statements that contain a [GROUP BY](#) aggregate: [CUBE](#), [GROUPING SETS](#), and [ROLLUP](#).

Behavior Type

Immutable

Syntax

GROUPING_ID ([*expression* [, ...])

<i>expression</i>	An expression that matches one of the expressions in the GROUP BY clause. If the GROUP BY clause includes a list of expressions, GROUPING_ID returns a number corresponding to the GROUPING bit vector associated with a row.
-------------------	--

Examples

This example shows how calling `GROUPING_ID` without an expression returns the `GROUPING` bit vector associated with a full set of multilevel aggregate expressions. The `GROUPING_ID` value is comparable to `GROUPING_ID(a,b)` because `GROUPING_ID()` includes all columns in the `GROUP BY ROLLUP`:

```
=> SELECT a,b,COUNT(*), GROUPING_ID() FROM T GROUP BY ROLLUP(a,b);
```

In the following query, the `GROUPING(Category)` and `GROUPING(Year)` columns have three combinations:

- 0,0
- 0,1
- 1,1

```
=> SELECT Category, Year, SUM(Amount),
       GROUPING(Category), GROUPING(Year) FROM expenses
       GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

Category	Year	SUM	GROUPING	GROUPING
Books	2005	39.98	0	0
Books	2007	29.99	0	0
Books	2008	29.99	0	0
Books		99.96	0	1
Electricity	2005	109.99	0	0
Electricity	2006	109.99	0	0
Electricity	2007	229.98	0	0
Electricity		449.96	0	1
		549.92	1	1

`GROUPING_ID` converts these values as follows:

Binary Set Values	Decimal Equivalents
00	0
01	1
11	3
0	Category, Year

The following query returns the single number for each `GROUP BY` level that appears in the `gr_id` column:

```
=> SELECT Category, Year, SUM(Amount),
        GROUPING(Category),GROUPING(Year),GROUPING_ID(Category,Year) AS gr_id
        FROM expenses GROUP BY ROLLUP(Category, Year);
Category | Year | SUM | GROUPING | GROUPING | gr_id
-----+-----+-----+-----+-----+-----
Books    | 2008 | 29.99 | 0 | 0 | 0
Books    | 2005 | 39.98 | 0 | 0 | 0
Electricity | 2007 | 229.98 | 0 | 0 | 0
Books    | 2007 | 29.99 | 0 | 0 | 0
Electricity | 2005 | 109.99 | 0 | 0 | 0
Electricity |      | 449.96 | 0 | 1 | 1
          |      | 549.92 | 1 | 1 | 3
Electricity | 2006 | 109.99 | 0 | 0 | 0
Books    |      | 99.96 | 0 | 1 | 1
```

The `gr_id` value determines the `GROUP BY` level for each row:

GROUP BY Level	GROUP BY Row Level
3	Total sum
1	Category
0	Category, year

You can also use the [DECODE](#) function to give the values more meaning by comparing each search value individually:

```
=> SELECT Category, Year, SUM(AMOUNT), DECODE(GROUPING_ID(Category, Year),
        3, 'Total',
        1, 'Category',
        0, 'Category,Year')
        AS GROUP_NAME FROM expenses GROUP BY ROLLUP(Category, Year);
Category | Year | SUM | GROUP_NAME
-----+-----+-----+-----
Electricity | 2006 | 109.99 | Category,Year
Books      |      | 99.96 | Category
Electricity | 2007 | 229.98 | Category,Year
Books      | 2007 | 29.99 | Category,Year
Electricity | 2005 | 109.99 | Category,Year
Electricity |      | 449.96 | Category
          |      | 549.92 | Total
Books      | 2005 | 39.98 | Category,Year
Books      | 2008 | 29.99 | Category,Year
```

See Also

- [CUBE Aggregate](#)
- [GROUP_ID](#)
- [GROUPING](#)
- [GROUPING SETS Aggregate](#)
- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

CUBE Aggregate

Automatically performs all possible aggregations of the specified columns, as an extension to the [GROUP BY](#) clause.

You can use the ROLLUP clause with three grouping functions:

- [GROUPING](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)

Syntax

GROUP BY *group-expression*[, ...]

Parameters

<i>group-expression</i>	<p>One or both of the following:</p> <ul style="list-style-type: none">• An expression that is not an aggregate or a grouping function that includes constants and column references in FROM-specified tables. For example: <p><code>column1, (column2+1), column3+column4</code></p>
-------------------------	---

	<ul style="list-style-type: none">• A multilevel expression, one of the following:<ul style="list-style-type: none">■ ROLLUP■ CUBE■ GROUPING SETS
--	---

Restrictions

- GROUP BY CUBE does not order data. If you want to sort data, use the [ORDER BY Clause](#). The ORDER BY clause must come *after* the GROUP BY clause.
- You can use CUBE inside a GROUPING SETS expression, but not inside a ROLLUP expression or another CUBE expression.

Levels of CUBE Aggregation

If n is the number of grouping columns, CUBE creates 2^n levels of aggregations. For example:

CUBE (A, B, C) creates all possible permutations, resulting in eight groups:

- (A, B, C)
- (A, B)
- (A, C)
- (B, C)
- (A)
- (B)
- (C)
- ()

If you increase the number of CUBE columns, the number of CUBE groupings increases exponentially. The CUBE query may be resource intensive and produce combinations that are not of interest. In that case, consider using the [GROUPING SETS Aggregate](#), which allows you to choose specific groupings.

Example: Using CUBE to Return All Groupings

Suppose you have a table that contains information about family expenses for books and electricity:

```
=> SELECT * FROM expenses ORDER BY Category, Year;
Year | Category | Amount
-----+-----+-----
2005 | Books    | 39.98
2007 | Books    | 29.99
2008 | Books    | 29.99
2005 | Electricity | 109.99
2006 | Electricity | 109.99
2007 | Electricity | 229.98
```

To aggregate the data by both Category and Year using the CUBE aggregate:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY CUBE(Category, Year) ORDER BY 1, 2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Books    |      | 99.96
Electricity | 2005 | 109.99
Electricity | 2006 | 109.99
Electricity | 2007 | 229.98
Electricity |      | 449.96
          | 2005 | 149.97
          | 2006 | 109.99
          | 2007 | 259.97
          | 2008 | 29.99
          |      | 549.92
```

The results include subtotals for each category and year, and a grand total (\$549.92).

Example: Using CUBE with the HAVING Clause

This example shows how you can restrict the GROUP BY results, use the HAVING clause with the CUBE aggregate. This query returns only the category totals and the full total:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY CUBE(Category,Year) HAVING GROUPING(Year)=1;
Category | Year | SUM
-----+-----+-----
Books    |      | 99.96
Electricity |      | 449.96
          |      | 549.92
```

The next query returns only the aggregations for the two categories for each year. The GROUPING ID function specifies to omit the grand total (\$549.92):

```
=> SELECT Category, Year, SUM (Amount) FROM expenses
      GROUP BY CUBE(Category,Year) HAVING GROUPING_ID(Category,Year)<2
      ORDER BY 1, 2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Books    |      | 99.96
Electrical | 2005 | 109.99
Electrical | 2006 | 109.99
Electrical | 2007 | 229.98
Electrical |      | 449.96
```

See Also

- [Data Aggregation](#)
- [GROUP BY Clause](#)
- [GROUP_ID](#)
- [GROUPING](#)
- [GROUPING_ID](#)
- [GROUPING SETS Aggregate](#)
- [ROLLUP Aggregate](#)

GROUPING SETS Aggregate

The `GROUPING SETS` aggregate is an extension to the `GROUP BY` clause that automatically performs subtotal aggregations on groupings that you specify.

You can use the `GROUPING SETS` clause with three grouping functions:

- [GROUPING](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)

To sort data, use the `ORDER BY` clause. The `ORDER BY` clause must follow the `GROUP BY` clause.

Syntax

GROUP BY *group-expression*[, ...]

Parameters

<i>group-expression</i>	<p>One or both of the following:</p> <ul style="list-style-type: none"> • An expression that is not an aggregate or a grouping function that includes constants and column references in FROM-specified tables. For example: <code>column1, (column2+1), column3+column4</code> • A multilevel expression, one of the following: <ul style="list-style-type: none"> ■ ROLLUP ■ CUBE ■ GROUPING SETS
-------------------------	---

Defining the Groupings

GROUPING SETS allows you to specify exactly which groupings you want in the results. You can also concatenate the groupings as follows:

The following example clauses result in the groupings shown.

This clause...	Defines groupings...
<code>...GROUP BY GROUPING SETS(A,B,C,D)...</code>	(A), (B), (C), (D)
<code>...GROUP BY GROUPING SETS((A),(B),(C),(D))...</code>	(A), (B), (C), (D)
<code>...GROUP BY GROUPING SETS((A,B,C,D))...</code>	(A, B, C, D)
<code>...GROUP BY GROUPING SETS(A,B),GROUPING SETS(C,D)...</code>	(A, C), (B, C), (A, D), (B, C)
<code>...GROUP BY GROUPING SETS((A,B)),GROUPING SETS(C,D)...</code>	(A, B, C), (A, B, D)
<code>...GROUP BY GROUPING SETS(A,B),GROUPING SETS(ROLLUP(C,D))...</code>	(A,B), (A,B,C), (A,B,C,D)

This clause...	Defines groupings...
<pre>...GROUP BY A,B,C, GROUPING SETS(ROLLUP(C, D))...</pre>	<p>(A, B, C, D), (A, B, C), (A, B, C)</p> <p>The clause contains two groups (A, B, C). In the HAVING clause, use the GROUP_ID function as a predicate, to eliminate the second grouping.</p>

Example: Selecting Groupings

This example shows how to select only those groupings you want. Suppose you want to aggregate on columns only, and you do not need the grand total. The first query omits the total. In the second query, you add () to the GROUPING SETS list to get the total. Use the ORDER BY clause to sort the results by grouping:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY GROUPING SETS((Category, Year), (Year))
      ORDER BY 1, 2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Electrical | 2005 | 109.99
Electrical | 2006 | 109.99
Electrical | 2007 | 229.98
          | 2005 | 149.97
          | 2006 | 109.99
          | 2007 | 259.97
          | 2008 | 29.99

=> SELECT Category, Year, SUM(Amount) FROM expenses
      GROUP BY GROUPING SETS((Category, Year), (Year), ())
      ORDER BY 1, 2, GROUPING_ID();
Category | Year | SUM
-----+-----+-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Electrical | 2005 | 109.99
Electrical | 2006 | 109.99
Electrical | 2007 | 229.98
          | 2005 | 149.97
          | 2006 | 109.99
          | 2007 | 259.97
          | 2008 | 29.99
          |      | 549.92
```

See Also

- [Data Aggregation](#)
- [CUBE Aggregate](#)
- [GROUPING](#)
- [GROUP_ID](#)
- [GROUPING_ID](#)
- [GROUP BY Clause](#)
- [ROLLUP Aggregate](#)

HAVING Clause

Restricts the results of a [GROUP BY Clause](#).

Syntax

```
HAVING condition [, ...]
```

Parameters

<i>condition</i>	Must unambiguously reference a grouping column, unless the reference appears within an aggregate function
------------------	---

Notes

- Semantically, the HAVING clause occurs after the GROUP BY operation.
- You can use expressions in the HAVING clause.
- The HAVING clause was added to the SQL standard because you cannot use WHERE with [Aggregate Functions](#).

Example

The following example returns the employees with salaries greater than \$50,000:

```
=> SELECT employee_last_name, MAX(annual_salary) as "highest_salary"
   FROM employee_dimension
   GROUP BY employee_last_name
   HAVING MAX(annual_salary) > 50000;
employee_last_name | highest_salary
-----+-----
Bauer              |          920149
Brown              |          569079
Campbell           |          649998
Carcetti           |          195175
Dobisz             |          840902
Farmer             |          804890
Fortin             |          481490
Garcia             |          811231
Garnett            |          963104
Gauthier           |          927335
(10 rows)
```

INTERSECT Clause

Calculates the intersection of the results of two or more SELECT queries. INTERSECT returns distinct values by both the query on the left and right sides of the INTERSECT operand.

Syntax

```
select
... INTERSECT select
... [ INTERSECT select ]...
... [ ORDER BY { column-name | ordinal-number } [ ASC | DESC ]
... [ , ... ] ]
... [ LIMIT { integer | ALL } ]
... [ OFFSET integer ]
```

Notes

- Use the INTERSECT clause to return all elements that are common to the results of all the SELECT queries. The INTERSECT query operates on the results of two or more SELECT queries. INTERSECT returns only the rows that are returned by all the specified queries.
- You cannot use the ALL keyword with an INTERSECT query.

- The results of each SELECT query must be union compatible; they must return the same number of columns, and the corresponding columns must have compatible data types. For example, you cannot use the INTERSECT clause on a column of type INTEGER and a column of type VARCHAR. If the SELECT queries do not meet these criteria, Vertica returns an error.

Note: The [Data Type Coercion Chart](#) lists the data types that can be cast to other data types. If one data type can be cast to the other, those two data types are compatible.

- Order the results of an INTERSECT operation by using an ORDER BY clause. In the ORDER BY list, specify the column names from the leftmost SELECT statement or specify integers that indicate the position of the columns by which to sort.
- You can use INTERSECT in FROM, WHERE, and HAVING clauses.
- The rightmost ORDER BY, LIMIT, or OFFSET clauses in an INTERSECT query do not need to be enclosed in parentheses because the rightmost query specifies that Vertica perform the operation on the results of the INTERSECT operation. Any ORDER BY, LIMIT, or OFFSET clauses contained in SELECT queries that appear earlier in the INTERSECT query must be enclosed in parentheses.
- The order by column names is from the first select.
- Vertica supports INTERSECT noncorrelated subquery predicates. For example:

```
=> SELECT * FROM T1
    WHERE T1.x IN
      (SELECT MAX(c1) FROM T2
       INTERSECT
       SELECT MAX(cc1) FROM T3
       INTERSECT
       SELECT MAX(d1) FROM T4);
```

Examples

Consider the following three tables:

Company_A

id	emp_lname	dept	sales
1234	Stephen	auto parts	1000
5678	Alice	auto parts	2500
9012	Katherine	floral	500
3214	Smithson	sporting goods	1500

Company_B

id	emp_lname	dept	sales
4321	Marvin	home goods	250
9012	Katherine	home goods	500
8765	Bob	electronics	20000
3214	Smithson	home goods	1500

Company_C

id	emp_lname	dept	sales
3214	Smithson	sporting goods	1500
5432	Madison	sporting goods	400
7865	Cleveland	outdoor	1500
1234	Stephen	floral	1000

The following query returns the IDs and last names of employees that exist in both Company_A and Company_C:

```
=> SELECT id, emp_lname FROM Company_A
      INTERSECT
      SELECT id, emp_lname FROM Company_C;
id  | emp_lname
-----+-----
3214 | Smithson
9012 | Katherine
(2 rows)
```

The following query returns the same two employees in descending order of sales:

```
=> SELECT id, emp_lname, sales FROM Company_A
      INTERSECT
      SELECT id, emp_lname, sales FROM Company_B
      ORDER BY sales DESC;
id  | emp_lname | sales
-----+-----+-----
3214 | Smithson  | 1500
9012 | Katherine | 500
(2 rows)
```

You can also use the integer that represents the position of the sales column (3) to return the same result:

```
=> SELECT id, emp_lname, sales FROM Company_A
      INTERSECT
      SELECT id, emp_lname, sales FROM Company_B
      ORDER BY 3 DESC;
id  | emp_lname | sales
-----+-----+-----
3214 | Smithson  | 1500
9012 | Katherine | 500
(2 rows)
```

The following query returns the employee who works for both companies whose sales in Company_B are greater than 1000:

```
=> SELECT id, emp_lname, sales FROM Company_A
      INTERSECT
      (SELECT id, emp_lname, sales FROM company_B WHERE sales > 1000)
      ORDER BY sales DESC;
 id | emp_lname | sales
-----+-----+-----
3214 | Smithson  | 1500
(1 row)
```

In the following query returns the ID and last name of the employee who works for all three companies:

```
=> SELECT id, emp_lname FROM Company_A
      INTERSECT
      SELECT id, emp_lname FROM Company_B
      INTERSECT
      SELECT id, emp_lname FROM Company_C;
 id | emp_lname
-----+-----
3214 | Smithson
(1 row)
```

The following query shows the results of a mismatched data types; these two queries are not union compatible:

```
=> SELECT id, emp_lname FROM Company_A
      INTERSECT
      SELECT emp_lname, id FROM Company_B;
ERROR 3429: For 'INTERSECT', types int and varchar are inconsistent
DETAIL: Columns: id and emp_lname
```

Using the [VMart](#) example database, the following query returns information about all Connecticut-based customers who bought items online and whose purchase amounts were between \$400 and \$500:

```
=> SELECT customer_key, customer_name from public.customer_dimension
      WHERE customer_key IN (SELECT customer_key
                             FROM online_sales.online_sales_fact
                             WHERE sales_dollar_amount > 400
                             INTERSECT
                             SELECT customer_key FROM online_sales.online_sales_fact
                             WHERE sales_dollar_amount < 500)
      AND customer_state = 'CT';
 customer_key | customer_name
-----+-----
39 | Sarah S. Winkler
44 | Meghan H. Overstreet
70 | Jack X. Cleveland
103 | Alexandra I. Vu
110 | Matt . Farmer
173 | Mary R. Reyes
```

```
188 | Steve G. Williams  
233 | Theodore V. McNulty  
250 | Marcus E. Williams  
294 | Samantha V. Young  
313 | Meghan P. Pavlov  
375 | Sally N. Vu  
384 | Emily R. Smith  
387 | Emily L. Garcia
```

...

The previous query returns the same data as:

```
=> SELECT customer_key,customer_name FROM public.customer_dimension  
      WHERE customer_key IN (SELECT customer_key  
                             FROM online_sales.online_sales_fact  
                             WHERE sales_dollar_amount > 400  
                             AND sales_dollar_amount < 500)  
      AND customer_state = 'CT';
```

See Also

- [SELECT](#)
- [EXCEPT Clause](#)
- [UNION Clause](#)
- [Subqueries](#)

INTO TABLE Clause

Creates a table from a query result set.

Syntax

Permanent table

```
INTO [TABLE] table-name
```

Temporary table

```
INTO [scope] TEMP[ORARY] [TABLE] table-name  
... [ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

Parameters

<p><i>scope</i></p>	<p>Specifies visibility of a temporary table definition:</p> <ul style="list-style-type: none"> • GLOBAL (default): The table definition is visible to all sessions, and persists until you explicitly drop the table. • LOCAL: The table definition is visible only to the session in which it is created, and is dropped when the session ends. <p>Regardless of this setting, retention of temporary table data is set by the keywords ON COMMIT DELETE ROWS and ON COMMIT PRESERVE ROWS (see below).</p> <p>For more information, see Creating Temporary Tables in the Administrator's Guide.</p>
<p>[TABLE] <i>table-name</i></p>	<p>Specifies the name of the table to create.</p>
<p>ON COMMIT { PRESERVE DELETE } ROWS</p>	<p>Specifies whether data is transaction- or session-scoped:</p> <ul style="list-style-type: none"> • DELETE (default) marks the temporary table for transaction-scoped data. Vertica removes all table data after each commit. • PRESERVE marks the temporary table for session-scoped data, which is preserved beyond the lifetime of a single transaction. Vertica removes all table data when the session ends.

Examples

The following SELECT statement has an INTO TABLE clause that creates table `newTable` from `customer_dimension`:

```
=> SELECT * INTO TABLE newTable FROM customer_dimension;
```

The following `SELECT` statement creates temporary table `newTempTable`. By default, temporary tables are created at a global scope, so its definition is visible to other sessions and persists until it is explicitly dropped. No `customer_dimension` data is copied into the new table, and Vertica issues a warning accordingly:

```
=> SELECT * INTO TEMP TABLE newTempTable FROM customer_dimension;
WARNING 4102: No rows are inserted into table "public"."newTempTable" because ON COMMIT DELETE ROWS
is the default for create temporary table
HINT: Use "ON COMMIT PRESERVE ROWS" to preserve the data in temporary table
CREATE TABLE
```

The following `SELECT` statement creates local temporary table `newTempTableLocal`. This table is visible only to the session in which it was created, and is automatically dropped when the session ends. The `INTO TABLE` clause includes `ON COMMIT PRESERVE ROWS`, so Vertica copies all selection data into the new table:

```
=> SELECT * INTO LOCAL TEMP TABLE newTempTableLocal ON COMMIT PRESERVE ROWS
      FROM customer_dimension;
CREATE TABLE
```

LIMIT Clause

Specifies the maximum number of result set rows to return.

Syntax

```
LIMIT { rows | ALL }
```

Parameters

<i>rows</i>	The maximum number of rows to return.
ALL	(default) Returns all rows.

Dependencies

- Use an [ORDER BY clause](#) with `LIMIT`. Otherwise, the query returns an undefined subset of the result set. For example, the following `SELECT` statement omits `ORDER BY`. Successive iterations of this query are liable to return a different set of five records from the

customer_dimension table:

```
=> SELECT customer_name, customer_city FROM customer_dimension LIMIT 5;
  customer_name | customer_city
-----+-----
Craig S. Robinson | Fayetteville
Mark M. Kramer   | Joliet
Barbara S. Farmer | Alexandria
Julie S. McNulty | Grand Prairie
Meghan R. Garcia | Athens
(5 rows)
```

In contrast, the following SELECT statement includes an ORDER BY clause and returns a consistent set of results:

```
=> SELECT customer_name, customer_city FROM customer_dimension
  ORDER BY customer_city, customer_name LIMIT 5;
  customer_name | customer_city
-----+-----
Alexander . Dobisz | Abilene
Alexander B. McCabe | Abilene
Alexander J. Goldberg | Abilene
Alexander M. Fortin | Abilene
Alexander P. Moore | Abilene
(5 rows)
```

- LIMIT must follow the SELECT statement's [ORDER BY clause](#).
- When a SELECT statement specifies both LIMIT and [OFFSET](#), Vertica first processes the OFFSET statement, and then applies the LIMIT statement to the remaining rows.

MATCH Clause

A SQL extension that lets you screen large amounts of historical data in search of event patterns, the MATCH clause provides subclasses for analytic partitioning and ordering and matches rows from the result table based on a pattern you define.

You specify a pattern as a regular expression, which is composed of event types defined in the DEFINE subclause, where each event corresponds to a row in the input table. Then you can search for the pattern within a sequence of input events. Pattern matching returns the contiguous sequence of rows that conforms to PATTERN subclause. For example, pattern P (A B* C) consist of three event types: A, B, and C. When Vertica finds a match in the input table, the associated pattern instance must be an event of type A followed by 0 or more events of type B, and an event of type C.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- Landed on the company home page
- Navigated to the product page
- Ran a search
- Clicked a link from the search results
- Made a purchase

For examples that use this clickstream model, see [Event Series Pattern Matching](#) in Analyzing Data.

Syntax

```
MATCH ( [ PARTITION BY table_column ] ORDER BY table_column  
... DEFINE event_name AS boolean_expr [, ...]  
... PATTERN pattern_name AS ( regexp )  
... [ ROWS MATCH { ALL EVENTS | FIRST EVENT } ] )
```

Parameters

PARTITION BY	[Optional] Defines the window data scope in which the pattern, defined in the PATTERN subclause, is matched. The partition clause partitions the data by matched patterns defined in the PATTERN subclause. For each partition, data is sorted by the ORDER BY clause. If the partition clause is omitted, the entire data set is considered a single partition.
ORDER BY	Defines the window data scope in which the pattern, defined in the PATTERN subclause, is matched. For each partition, the order clause specifies how the input data is ordered for pattern matching. Note: The ORDER BY clause is mandatory.

<p>DEFINE</p>	<p>Defines the boolean expressions that make up the event types in the regular expressions. For example:</p> <pre>DEFINE Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%', Onsite AS PageURL ILIKE '%website2.com%' AND Action='V', Purchase AS PageURL ILIKE '%website2.com%' AND Action='P'</pre> <p>The DEFINE subclause accepts a maximum of 52 events. See Event Series Pattern Matching in Machine Learning for Predictive Analytics for examples.</p>		
<p><i>event_name</i></p>	<p>Is the name of the event to evaluate for each row; for example, Entry, Onsite, Purchase.</p> <p>Note: Event names are case insensitive and follow the same naming conventions as those used for tables and columns.</p>		
<p><i>boolean_expr</i></p>	<p>Is an expression that returns true or false. <i>boolean_expr</i> can include Boolean Operators and relational (comparison) operators. For example:</p> <pre>Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'</pre>		
<p>PATTERN <i>pattern_name</i></p>	<p>Is the name of the pattern, which you define in the PATTERN subclause; for example, P is the pattern name defined below:</p> <pre>PATTERN P AS (...)</pre> <p>A PATTERN is a <i>search pattern</i> that is comprised of a name and a regular expression.</p> <p>Note: Vertica supports one pattern per query.</p>		
<p><i>regexp</i></p>	<p>Is a regular expression comprised of event types (defined in the DEFINE subclause), and one or more of the quantifiers below. When Vertica evaluates the MATCH clause, the regular expression identifies the rows that meet the expression criteria.</p> <table border="1" data-bbox="500 1759 1411 1829"> <tr> <td data-bbox="500 1759 594 1829"> <p>*</p> </td> <td data-bbox="594 1759 1411 1829"> <p>Match 0 or more times</p> </td> </tr> </table>	<p>*</p>	<p>Match 0 or more times</p>
<p>*</p>	<p>Match 0 or more times</p>		

	<table border="1"> <tr> <td>*?</td> <td>Match 0 or more times, not greedily</td> </tr> <tr> <td>+</td> <td>Match 1 or more times</td> </tr> <tr> <td>+?</td> <td>Match 1 or more times, not greedily</td> </tr> <tr> <td>?</td> <td>Match 0 or 1 time</td> </tr> <tr> <td>??</td> <td>Match 0 or 1 time, not greedily</td> </tr> <tr> <td>*+</td> <td>Match 0 or more times, possessive</td> </tr> <tr> <td>++</td> <td>Match 1 or more times, possessive</td> </tr> <tr> <td>?+</td> <td>Match 0 or 1 time, possessive</td> </tr> <tr> <td> </td> <td>Alternation. Matches expression before or after the vertical bar. Similar to a Boolean or.</td> </tr> </table>	*?	Match 0 or more times, not greedily	+	Match 1 or more times	+?	Match 1 or more times, not greedily	?	Match 0 or 1 time	??	Match 0 or 1 time, not greedily	*+	Match 0 or more times, possessive	++	Match 1 or more times, possessive	?+	Match 0 or 1 time, possessive		Alternation. Matches expression before or after the vertical bar. Similar to a Boolean or.
*?	Match 0 or more times, not greedily																		
+	Match 1 or more times																		
+?	Match 1 or more times, not greedily																		
?	Match 0 or 1 time																		
??	Match 0 or 1 time, not greedily																		
*+	Match 0 or more times, possessive																		
++	Match 1 or more times, possessive																		
?+	Match 0 or 1 time, possessive																		
	Alternation. Matches expression before or after the vertical bar. Similar to a Boolean or.																		
ROWS MATCH	<p>[Optional] Defines how to resolve more than one event evaluating to true for a single row.</p> <ul style="list-style-type: none"> If you use ROWS MATCH ALL EVENTS, Vertica returns the following run-time error if more than one event evaluates to true for a single row: <div data-bbox="537 1178 1401 1283" data-label="Code-Block" style="background-color: #f0f0f0; padding: 5px; margin: 10px 0;"> <pre>ERROR: pattern events must be mutually exclusive HINT: try using ROWS MATCH FIRST EVENT</pre> </div> For ROWS MATCH FIRST EVENT, if more than one event evaluates to true for a single row, Vertica chooses the event defined first in the SQL statement to be the event it uses for the row. 																		

Pattern Semantic Evaluation

- The semantic evaluating ordering of the SQL clauses is: FROM -> WHERE -> PATTERN MATCH -> SELECT.
- Data is partitioned as specified in the PARTITION BY clause. If the partition clause is omitted, the entire data set is considered a single partition.

- For each partition, the order clause specifies how the input data is ordered for pattern matching.
- Events are evaluated for each row. A row could have 0, 1, or *N* events evaluate to true. If more than one event evaluates to true for the same row, Vertica returns a run-time error unless you specify `ROWS MATCH FIRST EVENT`. If you specify `ROWS MATCH FIRST EVENT` and more than one event evaluates to `TRUE` for a single row, Vertica chooses the event that was defined first in the SQL statement to be the event it uses for the row.
- Vertica performs pattern matching by finding the contiguous sequence of rows that conforms to the pattern defined in the `PATTERN` subclause.

For each match, Vertica outputs the rows that contribute to the match. Rows not part of the match (do not satisfy one or more predicates) are not output.

- Vertica reports only non-overlapping matches. If an overlap occurs, Vertica chooses the first match found in the input stream. After finding the match, Vertica looks for the next match, starting at the end of the previous match.
- Vertica reports the longest possible match, not a subset of a match. For example, consider pattern: `A*B` with input: `AAAB`. Because `A` uses the greedy regular expression quantifier (`*`), Vertica reports all `A` inputs (`AAAB`), not `AAB`, `AB`, or `B`.

Notes and Restrictions

- `DISTINCT` and `GROUP BY/HAVING` clauses are not allowed in pattern match queries.
- The following expressions are not allowed in the `DEFINE` subclause:
 - Subqueries, such as `DEFINE X AS c IN (SELECT c FROM table1)`
 - Analytic functions, such as `DEFINE X AS c < LEAD(1) OVER (ORDER BY 1)`
 - Aggregate functions, such as `DEFINE X AS c < MAX(1)`
- You cannot use the same pattern name to define a different event; for example, the following is not allowed for `X`:

```
DEFINE X AS c1 < 3
X AS c1 >= 3
```

- Used with MATCH clause, Vertica [Pattern Matching Functions](#) provide additional data about the patterns it finds. For example, you can use the functions to return values representing the name of the event that matched the input row, the sequential number of the match, or a partition-wide unique identifier for the instance of the pattern that matched.

Examples

For examples, see [Event Series Pattern Matching](#) in Machine Learning for Predictive Analytics.

See Also

- [Pattern Matching Functions](#)
- [EVENT_NAME](#)
- [MATCH_ID](#)
- [PATTERN_ID](#)

MINUS Clause

MINUS is an alias for [EXCEPT](#).

OFFSET Clause

Omits a specified number of rows from the beginning of the result set.

Syntax

OFFSET *rows*

Parameters

<i>rows</i>	Specifies the number of result set rows to omit.
-------------	--

Dependencies

- Use an [ORDER BY clause](#) with OFFSET. Otherwise, the query returns an undefined subset of the result set.
- OFFSET must follow the [ORDER BY clause](#) in a SELECT statement or UNION clause.
- When a SELECT statement or UNION clause specifies both [LIMIT](#) and OFFSET, Vertica first processes the OFFSET statement, and then applies the LIMIT statement to the remaining rows.

Example

The following query returns 14 rows from the `customer_dimension` table:

```
=> SELECT customer_name, customer_gender FROM customer_dimension
      WHERE occupation='Dancer' AND customer_city = 'San Francisco' ORDER BY customer_name;
  customer_name | customer_gender
-----+-----
Amy X. Lang    | Female
Anna H. Li     | Female
Brian O. Weaver | Male
Craig O. Pavlov | Male
Doug Z. Goldberg | Male
Harold S. Jones | Male
Jack E. Perkins | Male
Joseph W. Overstreet | Male
Kevin . Campbell | Male
Raja Y. Wilson  | Male
Samantha O. Brown | Female
Steve H. Gauthier | Male
William . Nielson | Male
William Z. Roy  | Male
(14 rows)
```

If you modify the previous query to specify an offset of 8 (`OFFSET 8`) clause, Vertica skips the first eight rows of the previous result set. The query returns the following results:

```
=> SELECT customer_name, customer_gender FROM customer_dimension
      WHERE occupation='Dancer' AND customer_city = 'San Francisco' ORDER BY customer_name OFFSET 8;
  customer_name | customer_gender
-----+-----
Kevin . Campbell | Male
Raja Y. Wilson  | Male
Samantha O. Brown | Female
Steve H. Gauthier | Male
William . Nielson | Male
William Z. Roy  | Male
(6 rows)
```

ORDER BY Clause

Sorts a query result set on one or more columns.

Syntax

```
ORDER BY expression [ ASC | DESC ] [, ...]
```

Parameters

<i>expression</i>	One of the following: <ul style="list-style-type: none">Name or ordinal number of a SELECT list item. You cannot use an integer value for an ORDER BY clause that is inside an analytic function's OVER clause.Arbitrary expression formed from columns that do not appear in the SELECT listCASE expression
-------------------	--

Notes

- The ordinal number refers to the position of the result column, counting from the left beginning at one. This makes it possible to order by a column that does not have a unique name. (You can assign a name to a result column using the AS clause.)
- While the user's current locale and collation sequence are used to compare strings and determine the results of the ORDER BY clause of a query, Vertica projection data is always stored sorted by the ASCII (binary) collating sequence.
- For INTEGER, INT, and DATE/TIME data types, NULL appears first (smallest) in ascending order.
- For FLOAT, BOOLEAN, CHAR, and VARCHAR, NULL appears last (largest) in ascending order.
- The ORDER BY clause may contain only columns or expressions that are in the window partition clause (see [Window Partition Clause](#)).

Examples

The follow example returns all the city and deal size for customer Metamedia, sorted by deal size in descending order.

```
=> SELECT customer_city, deal_siz FROM customer_dimension WHERE customer_name = 'Metamedia'
      ORDER BY deal_size DESC;
-----
customer_city | deal_size
-----
El Monte      | 4479561
Athens        | 3815416
Ventura       | 3792937
Peoria        | 3227765
Arvada        | 2671849
Coral Springs | 2643674
Fontana       | 2374465
Rancho Cucamonga | 2214002
Wichita Falls | 2117962
Beaumont      | 1898295
Arvada        | 1321897
Waco          | 1026854
Joliet        | 945404
Hartford     | 445795
(14 rows)
```

The following example uses a transform function. It returns an error because the ORDER BY column is not in the window partition.

```
=> CREATE TABLE t(geom geometry(200), geog geography(200));
=> SELECT PolygonPoint(geom) OVER(PARTITION BY geom)
      AS SEL_0 FROM t ORDER BY geog;
ERROR 2521: Cannot specify anything other than user defined transforms and partitioning expressions
in the ORDER BY list
```

The following example, using the same table, corrects this error.

```
=> SELECT PolygonPoint(geom) OVER(PARTITION BY geom)
      AS SEL_0 FROM t ORDER BY geom;
```

TIMESERIES Clause

Provides gap-filling and interpolation (GFI) computation, an important component of time series analytics computation. See [Time Series Analytics](#) in Analyzing Data for details and examples.

Syntax

```
TIMESERIES slice-time AS 'length-and-time-unit-expr' OVER (
... [ PARTITION BY (column-expr[,...] )
... ORDER BY time-expr )
... [ ORDER BY table-column[, ... ] ]
```

Parameters

<i>slice-time</i>	<p>A time column produced by the TIMESERIES clause, which stores the time slice start times generated from gap filling.</p> <p>Note: This parameter is an alias, so you can use any name that an alias would take.</p>
<i>length-and-time-unit-expr</i>	<p>An INTERVAL DAY TO SECOND literal that specifies the length of time unit of time slice computation. For example:</p> <pre>TIMESERIES slice_time AS '3 seconds' ...</pre>
OVER()	<p>Specifies partitioning and ordering for the function. OVER() also specifies that the time series function operates on a query result set—that is, the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses are evaluated.</p>
PARTITION BY (<i>column-expr</i> [,...])	<p>Partitions the data by the specified column expressions. Gap filling and interpolation is performed on each partition separately</p>
ORDER BY <i>time-expr</i>	<p>Sorts the data by the TIMESTAMP expression <i>time-expr</i>, which computes the time information of the time series data.</p> <p>Note: The TIMESERIES clause requires an ORDER BY operation on the timestamp column.</p>

Notes

If the *window-partition-clause* is not specified in `TIMESERIES OVER()`, for each defined time slice, exactly one output record is produced; otherwise, one output record is produced per partition per time slice. Interpolation is computed there.

Given a query block that contains a `TIMESERIES` clause, the following are the semantic phases of execution (after evaluating the `FROM` and the optional `WHERE` clauses):

1. Compute *time-expression*.
2. Perform the same computation as the `TIME_SLICE()` function on each input record based on the result of *time-exp* and '*length-and-time-unit-expr*'.
 - a. Perform gap filling to generate time slices missing from the input.
 - b. Name the result of this computation as *slice_time*, which represents the generated “time series” column (alias) after gap filling.
3. Partition the data by *expression, slice-time*. For each partition, do step 4.
4. Sort the data by *time-expr*. Interpolation is computed here.

There is semantic overlap between the `TIMESERIES` clause and the [TIME_SLICE](#) function with the following key differences:

- `TIMESERIES` only supports the [interval qualifier](#) `DAY TO SECOND`; it does not allow `YEAR TO MONTH`.
- Unlike `TIME_SLICE`, the time slice length and time unit expressed in *length-and-time-unit-expr* must be constants so gaps in the time slices are well-defined.
- `TIMESERIES` performs gap filling; the `TIME_SLICE` function does not.
- `TIME_SLICE` can return the start or end time of a time slice, depending on the value of its fourth input parameter (*start-or-end*). `TIMESERIES`, on the other hand, always returns the start time of each time slice. To output the end time of each time slice, write a `SELECT` statement like the following:

```
=> SELECT slice_time + <slice_length>;
```

Restrictions

- When the `TIMESERIES` clause occurs in a SQL query block, only the following clauses can be used in the same query block:
 - `SELECT`
 - `FROM`
 - `WHERE`
 - `ORDER BY`
- `GROUP BY` and `HAVING` clauses are not allowed. If a `GROUP BY` operation is needed before or after gap-filling and interpolation (GFI), use a subquery and place the `GROUP BY` in the outer query. For example:

```
=> SELECT symbol, AVG(first_bid) as avg_bid FROM (  
    SELECT symbol, slice_time, TS_FIRST_VALUE(bid1) AS first_bid  
    FROM Tickstore  
    WHERE symbol IN ('MSFT', 'IBM')  
    TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol ORDER BY ts)  
    ) AS resultOfGFI  
GROUP BY symbol;
```

- When the `TIMESERIES` clause is present in the SQL query block, the `SELECT` list can include only the following:
 - Time series aggregate functions such as `TS_FIRST_VALUE` and `TS_LAST_VALUE`
 - `slice_time` column
 - `PARTITION BY` expressions
 - `TIME_SLICE` function

For example, the following two queries return a syntax error because `bid1` is not a `PARTITION BY` or `GROUP BY` column:

```
=> SELECT bid, symbol, TS_FIRST_VALUE(bid) FROM Tickstore  
    TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol ORDER BY ts);  
ERROR: column "Tickstore.bid" must appear in the PARTITION BY list of Timeseries clause or be  
used in a Timeseries Output function  
=> SELECT bid, symbol, AVG(bid) FROM Tickstore  
    GROUP BY symbol;  
ERROR: column "Tickstore.bid" must appear in the GROUP BY clause or be used in an aggregate  
function
```

Examples

For examples, see [Gap Filling and Interpolation \(GFI\)](#) in Analyzing Data.

See Also

- [TIME_SLICE](#)
- [TS_FIRST_VALUE](#)
- [TS_LAST_VALUE](#)
- [Gap Filling and Interpolation \(GFI\)](#)

UNION Clause

Combines the results of multiple SELECT statements. You can include UNION in [FROM](#), [WHERE](#), and [HAVING](#) clauses.

Syntax

```
select-stmt UNION { ALL | DISTINCT } select-stmt [ UNION { ALL | DISTINCT } select-stmt ]...  
... [ ORDER BY expression { ASC | DESC }[,...] ]  
... [ LIMIT { count | ALL } ]  
... [ OFFSET start ]
```

Parameters

<i>select-stmt</i>	<p>A SELECT statement that returns one or more rows, depending on whether you specify keywords DISTINCT or ALL.</p> <p>The following options also apply:</p> <ul style="list-style-type: none">• The first SELECT statement can include the hint LABEL. Vertica ignores LABEL hints in subsequent SELECT statements.• Each SELECT statement can specify its own ORDER BY, LIMIT, and OFFSET clauses. A SELECT statement with one or more of these clauses must be enclosed by parentheses. See also:
--------------------	---

	ORDER BY, LIMIT, and OFFSET Clauses in UNION.
DISTINCT ALL	<p>Specifies whether to return unique rows:</p> <ul style="list-style-type: none">• DISTINCT (default) returns only unique rows.• ALL concatenates all rows, including duplicates. For best performance, use UNION ALL.

Requirements

- All rows of the UNION result set must be in the result set of at least one of its SELECT statements.
- Each SELECT statement must specify the same number of columns.
- Data types of corresponding SELECT statement columns must be [compatible](#), otherwise Vertica returns an error.

ORDER BY, LIMIT, and OFFSET Clauses in UNION

A UNION statement can specify its own [ORDER BY](#), [LIMIT](#), and [OFFSET](#) clauses. For example, given the tables described below in [Examples](#), the following query orders the UNION result set by emp_name and limits output to the first two rows:

```
=> SELECT id, emp_name FROM company_a UNION ALL SELECT id, emp_name FROM company_b ORDER BY emp_name
LIMIT 2;
 id | emp_name
-----+-----
5678 | Alice
8765 | Bob
(2 rows)
```

Each SELECT statement in a UNION clause can specify its own [ORDER BY](#), [LIMIT](#), and [OFFSET](#) clauses. In this case, the SELECT statement must be enclosed by parentheses. Vertica processes the SELECT statement [ORDER BY](#), [LIMIT](#), and [OFFSET](#) clauses before it processes the UNION clauses.

For example, each SELECT statement in the following UNION specifies its own [ORDER BY](#) and [LIMIT](#) clauses. Vertica processes the individual queries and then concatenates the two result sets:

```
=> (SELECT id, emp_name FROM company_a ORDER BY emp_name LIMIT 2)
UNION ALL
(SELECT id, emp_name FROM company_b ORDER BY emp_name LIMIT 2);
```

```
id | emp_name
-----+-----
5678 | Alice
9012 | Katherine
8765 | Bob
9012 | Katherine
(4 rows)
```

The following requirements and restrictions determine how Vertica processes a UNION clause that contains [ORDER BY](#), [LIMIT](#), and [OFFSET](#) clauses:

- A UNION's ORDER BY clause must specify columns from the first (leftmost) SELECT statement.
- Always use an ORDER BY clause with LIMIT and OFFSET. Otherwise, the query returns an undefined subset of the result set.
- ORDER BY must precede LIMIT and OFFSET.
- When a SELECT or UNION statement specifies both LIMIT and OFFSET, Vertica first processes the OFFSET statement, and then applies the LIMIT statement to the remaining rows.

UNION in Non-Correlated Subqueries

Vertica supports UNION in [noncorrelated subquery predicates](#). For example:

```
=> SELECT DISTINCT customer_key, customer_name FROM public.customer_dimension WHERE customer_key IN
      (SELECT customer_key FROM store.store_sales_fact WHERE sales_dollar_amount > 500
       UNION ALL
       SELECT customer_key FROM online_sales.online_sales_fact WHERE sales_dollar_amount > 500)
      AND customer_state = 'CT';
customer_key | customer_name
-----+-----
7021 | Luigi T. Dobisz
1971 | Betty V. Dobisz
46284 | Ben C. Gauthier
33885 | Tanya Y. Taylor
5449 | Sarah O. Robinson
29059 | Sally Z. Fortin
11200 | Foodhope
15582 | John J. McNulty
24638 | Alexandra F. Jones
...
```

Examples

The examples that follow use these two tables:

company_a

ID	emp_name	dept	sales
1234	Stephen	auto parts	1000
5678	Alice	auto parts	2500
9012	Katherine	floral	500

company_b

ID	emp_name	dept	sales
4321	Marvin	home goods	250
9012	Katherine	home goods	500
8765	Bob	electronics	20000

Find all employee IDs and names from company_a and company_b

The UNION statement specifies DISTINCT to combine unique IDs and last names of employees; Katherine works for both companies, so she appears only once in the result set. DISTINCT is the default and can be omitted:

```
=> SELECT id, emp_name FROM company_a UNION DISTINCT SELECT id, emp_name FROM company_b ORDER BY id;
 id | emp_name
-----+-----
 1234 | Stephen
 4321 | Marvin
 5678 | Alice
 8765 | Bob
 9012 | Katherine
(5 rows)
```

The next UNION statement specifies the option ALL. Katherine works for both companies, so the query returns two records for her:

```
=> SELECT id, emp_name FROM company_a UNION ALL SELECT id, emp_name FROM company_b ORDER BY id;
 id | emp_name
-----+-----
 1234 | Stephen
 5678 | Alice
 9012 | Katherine
 4321 | Marvin
 9012 | Katherine
 8765 | Bob
(6 rows)
```

Find the top two top performing salespeople in each company

Each SELECT statement specifies its own ORDER BY and LIMIT clauses, so the UNION statement concatenates the result sets as returned by each query:

```
=> (SELECT id, emp_name, sales FROM company_a ORDER BY sales DESC LIMIT 2)
 UNION ALL
 (SELECT id, emp_name, sales FROM company_b ORDER BY sales DESC LIMIT 2);
 id | emp_name | sales
```

```
-----+-----+-----  
8765 | Bob      | 20000  
5678 | Alice    | 2500  
1234 | Stephen  | 1000  
9012 | Katherine | 500  
(4 rows)
```

Find all employee orders by sales

The UNION statement specifies its own ORDER BY clause, which Vertica applies to the entire result:

```
=> SELECT id, emp_name, sales FROM company_a  
UNION  
SELECT id, emp_name, sales FROM company_b  
ORDER BY sales;  
id | emp_name | sales  
-----+-----+-----  
4321 | Marvin    | 250  
9012 | Katherine | 500  
1234 | Stephen  | 1000  
5678 | Alice    | 2500  
8765 | Bob      | 20000  
(5 rows)
```

Calculate the sum of sales for each company grouped by department

Each SELECT statement has its own GROUP BY clause. UNION combines the aggregate results from each query:

```
=> (SELECT 'Company A' as company, dept, SUM(sales) FROM company_a  
GROUP BY dept)  
UNION  
(SELECT 'Company B' as company, dept, SUM(sales) FROM company_b  
GROUP BY dept)  
ORDER BY 1;  
company | dept | sum  
-----+-----+-----  
Company A | auto parts | 3500  
Company A | floral | 500  
Company B | electronics | 20000  
Company B | home goods | 750  
(4 rows)
```

See Also

- [SELECT](#)
- [EXCEPT Clause](#)

- [INTERSECT Clause](#)
- [Subqueries](#)

WHERE Clause

Eliminates rows from the result table that do not satisfy one or more predicates.

Syntax

```
WHERE boolean-expression [ subquery ] ...
```

Parameters

<i>boolean-expression</i>	Is an expression that returns true or false. Only rows for which the expression is true become part of the result set.
---------------------------	--

The *boolean-expression* can include [Boolean Operators](#) and the following elements:

- [BETWEEN-predicate](#)
- [Boolean-Predicate](#)
- [Column-Value-Predicate](#)
- [IN-predicate](#)
- [Join-Predicate](#)
- [LIKE-predicate](#)
- [NULL-predicate](#)

Notes

You can use parentheses to group expressions, predicates, and boolean operators. For example:

```
=> ... WHERE NOT (A=1 AND B=2) OR C=3;
```

Example

The following example returns the names of all customers in the Eastern region whose name starts with 'Amer'. Without the WHERE clause filter, the query returns *all* customer names in the customer_dimension table.

```
=> SELECT DISTINCT customer_name
   FROM customer_dimension
   WHERE customer_region = 'East'
   AND customer_name ILIKE 'Amer%';
customer_name
-----
Americare
Americom
Americore
Americorp
Ameridata
Amerigen
Amerihope
Amerimedia
Amerishop
Ameristar
Ameritech
(11 rows)
```

WITH Clause

WITH clauses are individually-evaluated SELECT statements for use in a larger container query. You can use WITH clauses to simplify complicated queries and reduce statement repetition.

WITH clauses are evaluated through inline expansion or (optionally) through materialization. For details, see [WITH Clauses in SELECT](#).

Syntax

The following syntax statement is illustrative, rather than syntactically exact, to show the possibility of numerous successive WITH queries in use with others:

```
WITH... with-query-1 [(col-name[,...])]AS (SELECT ...),
... with-query-2 [(col-name[,...])]AS (SELECT ...[with-query-1]),
.
.
.
... with-query-n [(col-name[,...])]AS (SELECT ...[with-query-1, with-query-2, with-query-n[,...]])
SELECT
.
.
.
```

Restrictions

- Each WITH clause query must be uniquely named. Same-name aliases for WITH clause query names return with an error.
- WITH clauses do not support INSERT, UPDATE, or DELETE statements.
- WITH clauses cannot be used recursively; they can only be specified in succession.

Examples

See [WITH Clauses in SELECT](#).

See Also

- [SELECT](#)
- [Subqueries](#)
- [WITH Clauses in SELECT](#)

SET DATESTYLE

Specifies how to format date/time output for the current session. Use [SHOW DATESTYLE](#) to verify the current output settings.

Syntax

```
SET DATESTYLE TO { arg | 'arg' }[, arg | 'arg' ]
```

Parameters

SET DATESTYLE has a single parameter, which can be set to one or two arguments that specify date ordering and style. Each argument can be specified singly or in combination with the other; if combined, they can be specified in any order.

The following table describes each style and the date ordering arguments it supports:

Date style arguments	Order arguments	Example
ISO (ISO 8601/SQL standard)	n/a	2016-03-16 00:00:00
GERMAN	n/a	16.03.2016 00:00:00
SQL	MDY (default)	03/16/2016 00:00:00
	DMY	16/03/2016 00:00:00
POSTGRES	MDY (default)	Wed Mar 16 00:00:00 2016
	DMY	Wed 16 Mar 00:00:00 2016

Vertica ignores the order argument for date styles ISO and GERMAN. If the date style is SQL or POSTGRES, the order setting determines whether dates are output in MDY or DMY order. Neither SQL nor POSTGRES support YMD order. If you specify YMD for SQL or POSTGRES, Vertica ignores it and uses their default MDY order.

Date styles and ordering can also affect how Vertica interprets input values. For more information, see [Date/Time Literals](#).

Privileges

None

Input Dependencies

In some cases, input format can determine output, regardless of date style and order settings:

- Vertica ISO output for DATESTYLE is ISO long form, but several input styles are accepted. If the year appears first in the input, YMD is used for input and output, regardless of the DATESTYLE value.
- [INTERVAL](#) input and output share the same format, with the following exceptions:
 - Units like CENTURY or WEEK are converted to years and days.
 - AGO is converted to the appropriate sign.

If the date style is set to ISO, output follows this format:

```
[ quantity unit [ ... ] ] [ days ] [ hours:minutes:seconds ]
```

Example

```
=> CREATE TABLE t(a DATETIME);
CREATE TABLE
=> INSERT INTO t values ('3/16/2016');
OUTPUT
-----
      1
(1 row)

=> SHOW DATESTYLE;
  name | setting
-----+-----
datestyle | ISO, MDY
(1 row)

=> SELECT * FROM t;
      a
-----
2016-03-16 00:00:00
(1 row)

=> SET DATESTYLE TO German;
SET
=> SHOW DATESTYLE;
  name | setting
-----+-----
datestyle | German, DMY
(1 row)

=> SELECT * FROM t;
      a
-----
16.03.2016 00:00:00
(1 row)

=> SET DATESTYLE TO SQL;
SET
=> SHOW DATESTYLE;
  name | setting
-----+-----
datestyle | SQL, DMY
(1 row)

=> SELECT * FROM t;
      a
-----
16/03/2016 00:00:00
(1 row)

=> SET DATESTYLE TO Postgres, MDY;
SET
=> SHOW DATESTYLE;
```

```
name | setting
-----+-----
datestyle | Postgres, MDY
(1 row)

=> SELECT * FROM t;
      a
-----
Wed Mar 16 00:00:00 2016
(1 row)
```

SET ESCAPE_STRING_WARNING

Issues a warning when a backslash is used in a string literal during the current session.

Syntax

```
SET ESCAPE_STRING_WARNING TO { ON | OFF }
```

Parameters

ON	[Default] Issues a warning when a back slash is used in a string literal. Tip: Organizations that have upgraded from earlier versions of Vertica can use this as a debugging tool for locating backslashes that used to be treated as escape characters, but are now treated as literals.
OFF	Ignores back slashes within string literals.

Privileges

No special permissions required.

Notes

- This statement works under vsql only.
- Turn off standard conforming strings before you turn on this parameter.

Tip: To set escape string warnings across all sessions, use the `EscapeStringWarnings` configuration parameter. See the [Internationalization Parameters](#) in the Administrator's Guide.

Examples

The following example shows how to turn OFF escape string warnings for the session.

```
=> SET ESCAPE_STRING_WARNING TO OFF;
```

See Also

- [SET STANDARD_CONFORMING_STRINGS](#)

SET INTERVALSTYLE

Specifies whether to include units in interval output for the current session.

Syntax

```
SET INTERVALSTYLE TO [ plain | units ]
```

Parameters

plain	Sets the default interval output to omit units. PLAIN is the default value.
units	Enables interval output to include subtype unit identifiers . When INTERVALSTYLE is set to units, the DATESTYLE parameter controls output. If you enable units and they do not display in the output, check the DATESTYLE parameter value, which must be set to ISO or POSTGRES for interval units to display.

Privileges

None

Examples

See [Setting Interval Unit Display](#).

SET LOCALE

Specifies locale for the current session.

Syntax

SET LOCALE TO *ICU-Locale-identifier*

Parameters

ICU-Locale-identifier

Specifies the ICU locale identifier to use, by default set to:

```
en_US@collation=binary
```

You can also use the `vsq` command `\locale` to set the current locale. An unqualified `\locale` command shows the current setting:

```
=> \locale
en_GB
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation=binary;
```

If set to null, Vertica sets locale to `en_US_POSIX`:

```
=> set locale to '';
INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
```

The following requirements apply:

- Vertica only supports the `COLLATION` keyword.

	<ul style="list-style-type: none">• Single quotes are mandatory to specify collation. <p>For a complete list of locale identifiers, see the ICU Project</p>
--	---

Privileges

None

Commonly Used Locales

de_DE	German (Germany)
en_GB	English (Great Britain)
es_ES	Spanish (Spain)
fr_FR	French (France)
pt_BR	Portuguese (Brazil)
pt_PT	Portuguese (Portugal)
ru_RU	Russian (Russia)
ja_JP	Japanese (Japan)
zh_CN	Chinese (China, simplified Han)
zh_Hant_TW	Chinese (Taiwan, traditional Han)

Examples

Set session locale to en_GB:

```
=> SET LOCALE TO en_GB;  
INFO 2567: Canonical locale: 'en_GB'  
Standard collation: 'LEN'  
English (United Kingdom)  
SET
```

Use the short form of a locale:

```
=> SET LOCALE TO LEN;  
INFO 2567: Canonical locale: 'en'  
Standard collation: 'LEN'  
English  
SET
```

Specify collation:

```
=> SET LOCALE TO 'tr_tr@collation=standard';  
INFO 2567: Canonical locale: 'tr_TR@collation=standard'  
Standard collation: 'LTR'  
Turkish (Turkey, collation=standard) Türkçe (Türkiye, Sıralama=standard)  
SET
```

See Also

- [Implement Locales for International Data Sets](#)
- [About Locale](#)

SET ROLE

Enables a role for the user's current session. The user can access privileges that have been granted to the role.

Note: If you set `EnableAllRolesOnLogin=1`, this eliminates the need for the user to run `SET ROLE <rolenames>` to enable any roles that have been granted to any user. For more information see [Security Parameters](#).

Syntax

```
SET ROLE { role [, ...] | NONE | ALL | ALL EXCEPT [, ...] | DEFAULT }
```

Parameters

role [, ...] | NONE | ALL | ALL EXCEPT [, ...] | DEFAULT

The name of one or more roles to set as the current role, or one of the following keywords:

	<ul style="list-style-type: none">• NONE disables all roles for the current user session.• ALL enables all of the roles to which the user has been granted access. Use GRANT (Role) to assign a role to a user.• ALL EXCEPT enable all the roles to which the user has access, with the exception of the role or roles indicated with this command.• DEFAULT sets the roles assigned to the user as the default roles.
--	---

Privileges

As a user, you can only set a role that has been granted to you. Use the [SHOW AVAILABLE ROLES](#) command to retrieve a list of the roles available to you.

Notes

- The DBADMIN user creates default roles that the DBADMIN then grants to a user. The DBADMIN can also grant a default role to a user using [ALTER USER](#).
- Enabling a role does not affect any other roles that are currently enabled. A user session can have more than one role enabled at a time. The user's permissions are the union of all the roles that are currently active, plus any permissions granted directly to the user.

Examples

This example shows the following:

- `SHOW AVAILABLE_ROLES;` lists the roles available to the user, but not enabled.
- `SET ROLE applogs;` enables the applogs role for the user.
- `SHOW ENABLED_ROLES;` lists the applogs role as enabled (SET) for the user.

- SET ROLE appuser; enables the appuser role for the user.
- SHOW ENABLED_ROLES now lists both applogs and appuser as enabled roles for the user.
- SET ROLE NONE disables all the users' enabled roles .
- SHOW ENABLED_ROLES shows that no roles are enabled for the user.

```
=> SHOW AVAILABLE_ROLES;
  name      |      setting
-----+-----
available roles | applogs, appadmin, appuser
(1 row)

=> SET ROLE applogs;
SET

=> SHOW ENABLED_ROLES;
  name      |      setting
-----+-----
enabled roles | applogs
(1 row)

=> SET ROLE appuser;
SET

=> SHOW ENABLED_ROLES;
  name      |      setting
-----+-----
enabled roles | applogs, appuser
(1 row)

=> SET ROLE NONE;SET

=> SHOW ENABLED_ROLES;
  name      |      setting
-----+-----
enabled roles |
(1 row)
```

Set User Default Roles

Though the DBADMIN user is normally responsible for setting a user's default roles, as a user you can set your own role. For example, if you run SET ROLE NONE all of your enabled roles are disabled. Then it was determined you need access to role1 as a default role. The DBADMIN uses [ALTER USER](#) to assign you a default role:

```
=> ALTER USER user1 default role role1;
```

This example sets role1 as user1's default role because the DBADMIN assigned this default role using ALTER USER.

```
user1 => SET ROLE default;
user1 => SHOW ENABLED_ROLES;
  name      | setting
-----|-----
enabled roles | role1
(1 row)
```

Set All Roles as Default

This example makes all roles granted to user1 default roles:

```
user1 => SET ROLE all;
user1 => show enabled roles;
  name      | setting
-----|-----
enabled roles | role1, role2, role3
(1 row)
```

Set All Roles as Default With EXCEPT

This example makes all the roles granted to the user default roles with the exception of role1.

```
user1 => set role all except role1;
user1 => SHOW ENABLED_ROLES
  name      | setting
-----|-----
enabled roles | role2, role3
(1 row)
```

SET SEARCH_PATH

Specifies the order in which Vertica searches schemas when a SQL statement specifies a table name that is unqualified by a schema name. `SET SEARCH_PATH` overrides the current session's search path, which is initially set from the user profile. This search path remains in effect until the next `SET SEARCH_PATH` statement, or the session ends. For details, see [Setting Search Paths](#) in the Administrator's Guide.

To view the current search path, use `SHOW SEARCH_PATH`.

Syntax

```
SET SEARCH_PATH { TO | = } { schema-List | DEFAULT }
```

Parameters

<i>schema-list</i>	<p>A comma-delimited list of schemas that indicates the order in which Vertica searches schemas for a table whose name is unqualified by a schema name.</p> <p>If the search path includes a schema that does not exist, or for which the user lacks access privileges, Vertica silently skips over that schema.</p>
DEFAULT	<p>Sets the search path to the database default:</p> <p>"\$user", public, v_catalog, v_monitor, v_internal</p>

Privileges

None

Examples

Show the current search path:

```
=> SHOW SEARCH_PATH;
  name      |                setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

Reset the search path to schemas store and public:

```
=> SET SEARCH_PATH TO store, public;
=> SHOW SEARCH_PATH;
  name      |                setting
-----+-----
search_path | store, public, v_catalog, v_monitor, v_internal
(1 row)
```

Reset the search path to the database default settings:

```
=> SET SEARCH_PATH TO DEFAULT;
SET
=> SHOW SEARCH_PATH;
  name      |                setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
```

(1 row)

SET SESSION AUTOCOMMIT

Sets whether statements automatically commit their transactions on completion. This statement is primarily used by the client drivers to enable and disable autocommit, you should never have to directly call it.

Syntax

```
SET SESSION AUTOCOMMIT TO { ON | OFF }
```

Parameters

ON	Enable autocommit. Statements automatically commit their transactions when they complete. This is the default setting for connections made using the Vertica client libraries.
OFF	Disable autocommit. Transactions are not automatically committed. This is the default for interactive sessions (connections made through vsql).

Privileges

No special permissions required.

Examples

This examples show how to set AUTOCOMMIT to 'on' and then to 'off'.

```
=> SET SESSION AUTOCOMMIT TO on;  
SET  
  
=> SET SESSION AUTOCOMMIT TO off;  
SET
```

See Also

- [Client Libraries](#)

SET SESSION CHARACTERISTICS AS TRANSACTION

Sets the isolation level and access mode of all transactions that start after this statement is issued.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the SERIALIZABLE isolation level to ensure consistency.

Syntax

```
SET SESSION CHARACTERISTICS AS TRANSACTION setting [, setting]
```

setting = one or both of the following:

- ISOLATION LEVEL *argument*
- READ ONLY | READ WRITE

ISOLATION LEVEL Arguments

The ISOLATION LEVEL clause determines what data the transaction can access when other transactions run concurrently. You cannot change the isolation level after the first query (SELECT) or DML statement (INSERT, DELETE, UPDATE) if a transaction has run.

Set ISOLATION LEVEL to one of the following arguments:

SERIALIZABLE	Sets the strictest level of SQL transaction isolation. This level emulates transactions serially, rather than concurrently. It holds locks and blocks write operations until the transaction completes.
--------------	---

	<p>Applications that use SERIALIZABLE must be prepared to retry transactions in the event of serialization failures. This isolation level is not recommended for normal query operations.</p> <p>Setting the transaction isolation level to SERIALIZABLE does not apply to temporary tables. Temporary tables are isolated by their transaction scope.</p>
REPEATABLE READ	Automatically converted to SERIALIZABLE.
READ COMMITTED	The default setting, allows concurrent transactions. Use READ COMMITTED isolation or snapshot isolation for normal query operations. See READ COMMITTED Versus Snapshot Isolation below for details on differences between them.
READ UNCOMMITTED	Automatically converted to READ COMMITTED.

READ WRITE/READ ONLY

You can set the transaction access mode with one of the following:

READ WRITE	Default
READ ONLY	<p>Disallows SQL statements that require write access:</p> <ul style="list-style-type: none"> • INSERT, UPDATE, DELETE, and COPY operations on any non-temporary table. • CREATE, ALTER, and DROP • GRANT, REVOKE • EXPLAIN if the SQL statement to explain requires write access. <p>Note: Setting the transaction session mode to read-only does not prevent all write operations.</p>

Privileges

None

Viewing Session Transaction Characteristics

`SHOW TRANSACTION_ISOLATION` and `SHOW TRANSACTION_READ_ONLY` show the transaction settings for the current session:

```
=> SHOW TRANSACTION_ISOLATION;
      name          | setting
-----+-----
transaction_isolation | SERIALIZABLE
(1 row)

=> SHOW TRANSACTION_READ_ONLY;
      name          | setting
-----+-----
transaction_read_only | true
(1 row)
```

READ COMMITTED Versus Snapshot Isolation

By itself, `AT EPOCH LATEST` produces purely historical query behavior. However, with `READ COMMITTED`, `SELECT` queries return the same result set as `AT EPOCH LATEST`, plus any changes made by the current transaction.

This is standard ANSI SQL semantics for ACID transactions. Any select query within a transaction sees the transaction's own changes regardless of isolation level.

SET SESSION GRACEPERIOD

Sets how long a session socket remains blocked while awaiting client input or output for a given query. If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated. If no grace period is set, the query can maintain its block on the socket indefinitely.

Vertica applies a session's grace period and `RUNTIMECAP` settings independently. If no grace period is set, a query can continue to block indefinitely on a session socket, regardless of the query's `RUNTIMECAP` setting.

Syntax

```
SET SESSION GRACEPERIOD duration
```

Parameters

<i>duration</i>	<p>Specifies how long a query can block on any session socket, one of the following:</p> <ul style="list-style-type: none">• '<i>interval</i>': Specifies as an interval the maximum grace period for current session queries, up to 20 days.• =DEFAULT: Sets the grace period for queries in this session to the user's GRACEPERIOD value. A new session is initially set to this value.• NONE: Valid only for superusers, removes any grace period previously set on session queries.
-----------------	---

Privileges

- Superusers can increase session grace period to any value, regardless of database or node settings.
- Non-superusers can only set the session grace period to a value equal to or lower than their own user setting. If no grace period is explicitly set for a user, the grace period for that user is inherited from the node or database settings.

Examples

See [Handling Session Socket Blocking](#) in the Administrator's Guide.

SET SESSION IDLESESSIONTIMEOUT

Sets the maximum amount of time that a session can remain idle before it exits.

Note: An idle session has no queries running.

Syntax

SET SESSION IDLESESSIONTIMEOUT *duration*

Parameters

<i>duration</i>	<p>Specifies the amount of time a session can remain idle before it exits:</p> <ul style="list-style-type: none">• NONE (default): No idle timeout set on the session.• '<i>interval</i>': Specifies as an interval the maximum amount of time a session can remain idle.• =DEFAULT: Sets the idle timeout period for this session to the user's IDLESESSIONTIMEOUT value.
-----------------	--

Privileges

- Superusers can increase the time a session can remain idle to any value, regardless of database or node settings.
- Non-superusers can only set the session idle time to a value equal to or lower than their own user setting. If no session idle time is explicitly set for a user, the session idle time for that user is inherited from the node or database settings.

Examples

See [Managing Client Connections](#) in the Administrator's Guide.

SET SESSION MEMORYCAP

Limits how much memory can be allocated to any request in this session. This limit only applies to the current session; it does not limit the total amount of memory used by multiple sessions.

Syntax

SET SESSION MEMORYCAP *Limit*

Parameters

<i>Limit</i>	<p>One of the following:</p> <ul style="list-style-type: none">• '<i>max-expression</i>': A string value that specifies the memory limit, one of the following:<ul style="list-style-type: none">■ <i>int</i>% — Expresses the maximum as a percentage of total memory available to the Resource Manager, where <i>int</i> is an integer value between 0 and 100. For example: MEMORYCAP '40%'■ <i>int</i>{K M G T} — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example: MEMORYCAP '10G'• =DEFAULT: Sets the memory cap for queries in this session to the user's MEMORYCAP value. A new session is initially set to this value.• NONE: Valid only for superusers, removes any grace period previously set on session queries.
--------------	---

Privileges

- Superusers can increase session memory cap to any value.
- Non-superusers can only set the session memory cap to a value equal to or lower than their own user setting.

Examples

Set the session memory cap to 2 gigabytes:

```
=> SET SESSION MEMORYCAP '2G';
SET
=> SHOW MEMORYCAP;
  name  | setting
-----+-----
memorycap | 2097152
(1 row)
```

Revert the memory cap to the default setting as specified in the user profile:

```
=> SET MEMORYCAP=DEFAULT;
SET
=> SHOW MEMORYCAP;
  name  | setting
-----+-----
memorycap | 2013336
(1 row)
```

See Also

[Managing Workloads](#)

SET SESSION MULTIPLEACTIVERESULTSETS

Enables or disable the execution of multiple active result sets (MARS) on a single JDBC connection. Using this option requires an active JDBC connection.

Syntax

```
SET SESSION MULTIPLEACTIVERESULTSETS TO { ON | OFF }
```

Parameters

ON	Enable MultipleActiveResultSets. Allows you to execute multiple result sets on a single connection.
OFF	Disable MultipleActiveResultSets. Allows only one active result set per connection. (Default value.)

Privileges

No special permissions required.

Examples

This example shows how you can set `MultipleActiveResultSets` to on and then to off:

```
=> SET SESSION MULTIPLEACTIVERESULTSETS TO on;  
SET  
  
=> SET SESSION MULTIPLEACTIVERESULTSETS TO off;  
SET
```

SET SESSION RESOURCE_POOL

Associates the user session with the specified resource pool.

Syntax

```
SET SESSION RESOURCE_POOL = { pool-name | DEFAULT }
```

Parameters

<i>pool-name</i>	The name of an existing resource pool to associate with the current session.
DEFAULT	Sets the session's resource pool to the user's default resource pool.

Privileges

- Superusers can assign their session to any available resource pool.
- Non-superusers must have `USAGE` privileges for the resource pool.

Examples

This example sets `ceo_pool` as the session resource pool:

```
=> SET SESSION RESOURCE_POOL = ceo_pool;  
SET
```

See Also

- [ALTER RESOURCE POOL](#)
- [CREATE RESOURCE POOL](#)
- [CREATE USER](#)
- [DROP RESOURCE POOL](#)
- [GRANT \(Resource Pool\)](#)
- [SET SESSION MEMORYCAP](#)
- [Managing Workloads](#)

SET SESSION RUNTIMECAP

Sets the maximum amount of time queries can run in a give session. If a query exceeds its session's `RUNTIMECAP` setting, Vertica terminates the query and returns an error. You cannot increase the `RUNTIMECAP` beyond the limit that is set in your [user profile](#).

Note: Vertica does not strictly enforce session `RUNTIMECAP` settings. If you time a query, you might discover that it runs longer than the `RUNTIMECAP` setting.

Syntax

```
SET SESSION RUNTIMECAP duration
```

Parameters

<i>duration</i>	<p>Specifies how long a given query can run in the current session, one of the following:</p> <ul style="list-style-type: none">• NONE (default): Removes a runtime limit for all current session queries.• '<i>interval</i>': Specifies as an interval the maximum runtime for current session queries, up to one year—for example, 1 minute or 100 seconds.• =DEFAULT: Sets maximum runtime for queries in this session to the user's RUNTIMECAP value.
-----------------	---

Privileges

- Superusers can increase session RUNTIMECAP to any value.
- Non-superusers can only set the session RUNTIMECAP to a value equal to or lower than their own user RUNTIMECAP.

Examples

Set the maximum query runtime for the current session to 10 minutes:

```
=> SET SESSION RUNTIMECAP '10 minutes';
```

Revert the session RUNTIMECAP to your user default setting:

```
=> SET SESSION RUNTIMECAP =DEFAULT;
SET
=> SHOW RUNTIMECAP;
  name  | setting
-----+-----
runtimecap | UNLIMITED
(1 row)
```

See Also

- [Setting a Runtime Limit for Queries](#)
- [Managing Workloads](#)

SET SESSION TEMPSPACECAP

Sets the maximum amount of temporary file storage space that any request issued by the session can consume.

Syntax

```
SET SESSION TEMPSPACECAP 'space-limit' / = default | NONE
```

Parameters

<i>'space-limit'</i>	<p>The maximum amount of temporary file space the session can use. To set a limit, use a numeric value followed by a unit (for example: '10G'). The unit can be one of the following:</p> <ul style="list-style-type: none">• % percentage of total temporary storage space available. (In this case, the numeric value must be 0-100).• K—Kilobytes• M—Megabytes• G—Gigabytes• T—Terabytes <p>Setting this value to = default sets the session's TEMPSPACECAP to the user's TEMPSPACECAP value.</p> <p>Setting this value to NONE results in the session having unlimited temporary storage space. This is the default value.</p>
----------------------	--

Privileges

- This command requires superuser privileges to increase the TEMPSPACECAP over the user's TEMPSPACECAP limit.
- Regular users can change the TEMPSPACECAP associated with their own sessions to any value less than or equal to their own TEMPSPACECAP. They cannot increase its value beyond their own TEMPSPACECAP value.

Notes

- This limit is per session, not per user. A user could open multiple sessions, each of which could use up to the TEMPSPACECAP.
- Any execution plan that exceeds its TEMPSPACECAP usage results in the error:

ERROR: Exceeded temp space cap.

Examples

The following command sets a TEMPSPACECAP of 20gigabytes on the session:

```
=> SET SESSION TEMPSPACECAP '20G';  
SET  
=> SHOW TEMPSPACECAP;  
   name      | setting  
-----+-----  
temp spacecap | 20971520  
(1 row)
```

Note: SHOW displays the TEMPSPACECAP in kilobytes.

To return the memorycap to the previous setting:

```
=> SET SESSION TEMPSPACECAP NONE;  
SET  
=> SHOW TEMPSPACECAP;  
   name      | setting  
-----+-----  
temp spacecap | UNLIMITED  
(1 row)
```

See Also

- [ALTER USER](#)
- [CREATE USER](#)
- [Managing Workloads](#)

SET STANDARD_CONFORMING_STRINGS

Treats backslashes as escape characters for the current session.

Syntax

```
SET STANDARD_CONFORMING_STRINGS TO { ON | OFF }
```

Parameters

ON	Makes ordinary string literals ('...') treat back slashes (\) literally. This means that back slashes are treated as string literals, not escape characters. (This is the default.)
OFF	Treats back slashes as escape characters.

Privileges

No special permissions required.

Notes

- This statement works under vsql only.
- When standard conforming strings are on, Vertica supports SQL:2008 string literals within Unicode escapes.
- Standard conforming strings must be ON to use Unicode-style string literals (U&' \nnnn ').

Tip: To set conforming strings across all sessions (permanently), use the `StandardConformingStrings` as described in [Internationalization Parameters](#) in the Administrator's Guide.

Examples

The following example shows how to turn off conforming strings for the session.

```
=> SET STANDARD_CONFORMING_STRINGS TO OFF;
```

The following command lets you verify the settings:

```
=> SHOW STANDARD_CONFORMING_STRINGS;
      name                | setting
-----+-----
standard_conforming_strings | off
(1 row)
```

The following example shows how to turn on conforming strings for the session.

```
=> SET STANDARD_CONFORMING_STRINGS TO ON;
```

See Also

- [SET ESCAPE_STRING_WARNING](#)

SET TIME ZONE

Changes the TIME ZONE run-time parameter for the current session. Use [SHOW TIMEZONE](#) to show the session's current time zone.

If you set the timezone using POSIX format, the timezone abbreviation you use overrides the default timezone abbreviation. If the [date style](#) is set to POSTGRES, the timezone abbreviation you use is also used when converting a timestamp to a string.

Syntax

```
SET TIME ZONE TO { value | 'value' }
```

Note: Vertica treats literals TIME ZONE and TIMEZONE as synonyms.

Parameters

<i>value</i>	<p>One of the following:</p> <ul style="list-style-type: none">• A time zone literal supported by Vertica. To view the default list of valid literals, see the files in the following directory: <code>opt/vertica/share/timezonesets</code>• A signed integer representing an offset from UTC in hours• An interval value• Constants LOCAL and DEFAULT, which respectively set the time zone to the one specified in environment variable TZ, or if TZ is undefined, to the operating system time zone.
--------------	---

Privileges

None

Examples

```
=> SET TIME ZONE TO DEFAULT;  
=> SET TIME ZONE TO 'PST8PDT'; -- Berkeley, California  
=> SET TIME ZONE TO 'Europe/Rome'; -- Italy  
=> SET TIME ZONE TO '-7'; -- UDT offset equivalent to PDT  
=> SET TIME ZONE TO INTERVAL '-08:00 HOURS';
```

See Also

[Using Time Zones With Vertica](#)

Time Zone Names for Setting TIME_ZONE

The following time zone names are recognized by Vertica as valid settings for the SQL time zone (the TIME_ZONE run-time parameter).

Note: The names listed here are for convenience only and might be out of date. Refer to the [Sources for Time Zone and Daylight Saving Time Data](#) page for precise information.

These names are not the same as the names shown in `/opt/vertica/share/timezonesets`, which are recognized by Vertica in date/time input values. The TIME_ZONE names shown below imply a local daylight-savings time rule, where date/time input names represent a fixed offset from UTC.

In many cases the same zone has several names. These are grouped together. The table is primarily sorted by commonly used zone names.

In addition to the names listed in the table, Vertica accepts time zone names of the form *STDoffset* or *STDoffsetDST*, where *STD* is a zone abbreviation, *offset* is a numeric offset in hours west from UTC, and *DST* is an optional daylight-savings zone abbreviation, assumed to stand for one hour ahead of the given offset. For example, if EST5EDT were not already a recognized zone name, it would be accepted and would be functionally equivalent to USA East Coast time. When a daylight-savings zone name is present, it is assumed to be used according to USA time zone rules, so this feature is of limited use outside North America. Be wary that this provision can lead to silently accepting bogus input, since there is no check on the reasonableness of the zone abbreviations. For example, `SET TIME_ZONE TO FOOBANKO` works, leaving the system effectively using a rather peculiar abbreviation for GMT.

Time Zone
Africa
America
Antarctica
Asia
Atlantic
Australia
CET
EET
Etc/GMT
Europe
Factory
Etc/GMT GMT GMT+0 GMT-0 GMT0 Greenwich Etc/Greenwich
Indian
MET
Pacific
UCT Etc UCT
UTC Universal Zulu Etc/UTC Etc/Universal

Time Zone
Etc/Zulu
WET

SHOW

Displays run-time parameters for the current session.

Syntax

```
SHOW { parameter | ALL }
```

Parameters

AUTOCOMMIT	Displays whether statements automatically commit their transactions when they complete.
AVAILABLE_ROLES	Lists all roles available to the user.
DATESTYLE	Displays the current style of date values. See SET DATESTYLE .
ENABLED_ROLES	Displays the roles enabled for the current session. See SET ROLE .
ESCAPE_STRING_WARNING	Displays whether warnings are issued when backslash escapes are found in strings. See SET ESCAPE_STRING_WARNING .
INTERVALSTYLE	Displays whether units are output when printing intervals. See SET INTERVALSTYLE .
LOCALE	Displays the current locale. See SET LOCALE .
MEMORYCAP	Displays the maximum amount of memory that any request use. See SET MEMORYCAP .

MULTIPLEACTIVERESULTSETS	Displays whether multiple active result sets on one connection are allowed. See SET SESSION MULTIPLEACTIVERESULTSETS
RESOURCE_POOL	Displays the resource pool that the session is using. See SET RESOURCE POOL .
RUNTIMECAP	Displays the maximum amount of time that queries can run in the session. See SET RUNTIMECAP
SEARCH_PATH	Displays the order in which Vertica searches schemas. See SET SEARCH_PATH .
STANDARD_CONFORMING_STRINGS	Displays whether backslash escapes are enabled for the session. See SET STANDARD_CONFORMING_STRINGS .
TEMPSPACECAP	Displays the maximum amount of temporary file space that queries can use in the session. See SET TEMPSPACECAP .
TIMEZONE	Displays the timezone set in the current session. See SET TIMEZONE .
TRANSACTION_ISOLATION	Displays the current transaction isolation setting, as described in SET SESSION CHARACTERISTICS AS TRANSACTION .
TRANSACTION_READ_ONLY	Displays the current read-only setting, as described in SET SESSION CHARACTERISTICS AS TRANSACTION .
ALL	Shows all run-time settings.

Privileges

None

Examples

Display all current runtime parameter settings:

```
=> SHOW ALL;
      name          |          setting
-----+-----
 locale             | en_US@collation=binary (LEN_KBINARY)
 autocommit         | off
 standard_conforming_strings | on
 escape_string_warning | on
 datestyle          | ISO, MDY
 intervalstyle      | plain
 timezone           | US/Eastern
 search_path        | "$user", public, v_catalog, v_monitor, v_internal
 transaction_isolation | READ COMMITTED
 transaction_read_only | false
 resource_pool      | general
 memorycap          | UNLIMITED
 tempstoragecap     | UNLIMITED
 runtimecap         | UNLIMITED
 enabled roles      |
 available roles    | applogs, appadmin
(15 rows)
```

Return current search path settings:

```
=> SHOW SEARCH_PATH;
      name          |          setting
-----+-----
 search_path        | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

Show the session's transaction isolation level:

```
=> SHOW TRANSACTION_ISOLATION;
      name          |          setting
-----+-----
 transaction_isolation | READ COMMITTED
(1 row)
```

Return the current transaction isolation level, as set by [SET SESSION CHARACTERISTICS AS TRANSACTION](#). False indicates that the default read/write setting is in effect:

```
=> SHOW TRANSACTION_READ_ONLY;
      name          |          setting
-----+-----
 transaction_read_only | false
(1 row)
```

To change to read only:

```
=> SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY;
```

The same SHOW command now returns true:

```
=> SHOW TRANSACTION_READ_ONLY;
      name          |          setting
-----+-----
```

```
-----+-----  
transaction_read_only | true  
(1 row)
```

SHOW CURRENT

Displays active configuration parameter values that are set at all levels. Vertica first checks values set at the session level. If a value is not set for a configuration parameter at the session level, Vertica next checks if the value is set for the node where you are logged in, and then checks the database level. If no values are set, `SHOW CURRENT` shows the default value for the configuration parameter. If the configuration parameter requires a restart to take effect, the active values shown might differ from the set values.

Syntax

```
SHOW CURRENT { parameter-name[,...] | ALL }
```

Parameters

<i>parameter-name</i>	Names of any configuration parameters you want to show.
ALL	Shows all configuration parameters set at all levels.

Privileges

- Superusers only
- Non-superusers: `SHOW CURRENT ALL` returns masked parameter settings. Attempts to view specific parameter settings return an error.

Examples

Show configuration parameters and their settings at all levels.

```
=> SHOW CURRENT ALL;
```

level	name	setting
DEFAULT	ActivePartitionCount	1
DEFAULT	AdvanceAHMInterval	180
DEFAULT	AHMBackupManagement	0
DATABASE	AnalyzeRowCountInterval	3600
SESSION	ForceUDxFencedMode	1
NODE	MaxClientSessions	0
...		

SHOW DATABASE

Displays configuration parameter values that are set for the database. If a configuration parameter is not set at the database level, `SHOW DATABASE` does not return a row for that parameter. If the configuration parameter requires a restart to take effect, the values shown might differ from the active values.

Syntax

```
SHOW DATABASE db-name { parameter-name[,...] | ALL }
```

Parameters

<i>db-name</i>	Name of database for which to show the parameter values.
<i>parameter-name</i>	Names of any configuration parameters you want to show.
ALL	Shows all configuration parameters set at the database level.

Privileges

- Superusers only
- Non-superusers: `SHOW DATABASE ALL` returns masked parameter settings. Attempts to view specific parameter settings return an error.

Examples

Show all configuration parameters that are set at the database level:

```
=> SHOW DATABASE VMart ALL;
      name          |          setting
-----+-----
AnalyzeRowCountInterval | 3600
DefaultSessionLocale   | en_US@collation=binary
JavaBinaryForUDx       | /usr/bin/java
(3 rows)
```

SHOW NODE

Displays configuration parameter values that are set for a node. If a configuration parameter is not set at the node level, `SHOW NODE` does not return a row for that parameter. If the configuration parameter requires a restart to take effect, the values shown might differ from the active values.

Syntax

```
SHOW NODE node_name { parameter_name [,...] | ALL }
```

Parameters

<i>node_name</i>	Name of node for which to show the parameter values.
<i>parameter_name</i>	Names of any configuration parameters you want to show.
ALL	Shows all configuration parameters that are set at the node level.

Privileges

- You must have superuser privileges to view some values. If you are not the superuser, and you use the the keyword 'ALL', parameter values are masked with ****. If the parameter is

specifically listed, an error message appears.

- Non-superusers cannot view database or node-level security parameters.

Examples

The following example shows how display all values set for v_vmart_node0001:

```
=> SHOW NODE v_vmart_node0001 ALL;  
name          | setting  
-----+-----  
MaxClientSessions | 0  
(1 row)
```

SHOW SESSION

Displays set configuration parameter values for the current session. If the configuration parameter is not set at the session level, `SHOW SESSION` does not return a row for that parameter. If the configuration parameter requires a restart to take effect, the values shown might differ from the active values.

Syntax

```
SHOW SESSION { parameter-name[,...] | ALL }
```

Parameters

<i>parameter-name</i>	Names of any configuration parameters you want to show.
ALL	Shows all configuration parameters set at the session level.

Privileges

- Superusers only
- Non-superusers: `SHOW CURRENT ALL` returns masked parameter settings. Attempts to view specific parameter settings return an error.

Examples

View all configuration parameters and their settings for the current session:

```
=> SHOW SESSION ALL:
name | setting
-----+-----
locale | en_US@collation=binary (LEN_KBINARY)
autocommit | off
standard_conforming_strings | on
escape_string_warning | on
datestyle | ISO, MDY
intervalstyle | plain
timezone | America/New_York
search_path | "$user", public, v_catalog, v_monitor, v_internal
transaction_isolation | READ COMMITTED
transaction_read_only | false
resource_pool | general
memorycap | UNLIMITED
tempstorage | UNLIMITED
runtimecap | UNLIMITED
enabled roles | dbduser*, dbadmin*, pseudosuperuser*
available roles | dbduser*, dbadmin*, pseudosuperuser*
ForceUDxFencedMode | 1
(17 rows)
```

START TRANSACTION

Starts a transaction block.

Syntax

```
START TRANSACTION [ isolation_level ]
```

where *isolation_level* is one of:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED } READ { ONLY | WRITE }
}
```

Parameters

Isolation level, described in the following table, determines what data the transaction can access when other transactions are running concurrently. The isolation level cannot be changed after the first query (SELECT) or DML statement (INSERT, DELETE, UPDATE) has run. A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the SERIALIZABLE isolation level to ensure consistency.

<pre>WORK TRANSACTION</pre>	<p>Have no effect; they are optional keywords for readability.</p>
<pre>ISOLATION LEVEL { SERIALIZABLE REPEATABLE READ READ COMMITTED READ UNCOMMITTED }</pre>	<ul style="list-style-type: none"> • SERIALIZABLE—Sets the strictest level of SQL transaction isolation. This level emulates transactions serially, rather than concurrently. It holds locks and blocks write operations until the transaction completes. Not recommended for normal query operations. • REPEATABLE READ—Automatically converted to SERIALIZABLE by Vertica. • READ COMMITTED (Default)—Allows concurrent transactions. Use READ COMMITTED isolation for normal query operations, but be aware that there is a subtle difference between them. See Transactions for more information. • READ UNCOMMITTED—Automatically converted to READ COMMITTED by Vertica.
<pre>READ {WRITE ONLY}</pre>	<p>Determines whether the transaction is read/write or read-only. Read/write is the default.</p> <p>Setting the transaction session mode to read-only disallows the following SQL commands, but does not prevent all disk write operations:</p> <ul style="list-style-type: none"> • INSERT, UPDATE, DELETE, and COPY if the table they would write to is not a temporary table • All CREATE, ALTER, and DROP commands

- GRANT, REVOKE, and EXPLAIN if the command it would run is among those listed.

Privileges

No special permissions required.

Notes

[BEGIN](#) performs the same function as `START TRANSACTION`.

Examples

This example shows how to start a transaction.

```
= > START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;  
START TRANSACTION  
  
=> CREATE TABLE sample_table (a INT);  
CREATE TABLE  
  
=> INSERT INTO sample_table (a) VALUES (1);  
OUTPUT  
-----  
1  
(1 row)
```

See Also

- [Transactions](#)
- [Creating Transactions](#)
- [COMMIT](#)
- [END](#)
- [ROLLBACK](#)

TRUNCATE TABLE

Removes all storage associated with a table, while leaving the table definition intact.

TRUNCATE TABLE auto-commits the current transaction after statement execution and cannot be rolled back.

TRUNCATE TABLE removes all table history preceding the current epoch, regardless of where that data resides (WOS or ROS) or how it is segmented. Immediately after TRUNCATE TABLE returns, AT EPOCH queries on the truncated table return nothing.

Syntax

```
TRUNCATE TABLE [[db-name.]schema.]table-name
```

Parameters

<code>[<i>db-name.</i>]schema</code>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDbObject</pre>
<code><i>table-name</i></code>	<p>The name of the anchor table or temporary table to truncate. You cannot truncate an external table.</p>

Privileges

One of the following privileges is required:

- Superuser
- Table owner
- User with USAGE privileges on the table's schema—see [GRANT \(Schema\)](#)
- User granted truncate privileges—see [GRANT \(Truncate\)](#)

A schema owner can drop a table but cannot truncate a table.

Examples

See [Truncating Tables](#) in the Administrator's Guide.

See Also

- [DELETE](#)
- [DROP TABLE](#)
- [Best Practices for DELETE and UPDATE](#)

UPDATE

Replaces the values of the specified columns in all rows for which a specific condition is true. All other columns and rows in the table are unchanged. If successful, UPDATE returns the number of rows updated. A count of 0 indicates no rows matched the condition.

UPDATE inserts new records into the WOS and marks the old records for deletion. If the WOS fills up, the operation overflows to the ROS.

Syntax

```
UPDATE [ /*+ hint[, hint] */ ] [schema.]table-reference [AS] alias  
... SET set-expression [, ... ]  
... [ FROM from-List ]  
... [ where-clause ]
```

Parameters

```
/*+ hint  
[, hint] */
```

One or both of the following hints:

- A load hint, one of the following: [AUTO](#), [DIRECT](#), or [TRICKLE](#)
- [LABEL](#)

<p><i>schema</i></p>	<p>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</p> <pre>myschema.thisDBObject</pre>
<p><i>table-reference</i></p>	<p>Specifies a table, one of the following:</p> <ul style="list-style-type: none"> • An optionally qualified table name with optional table aliases, column aliases, and outer joins. • An outer join table. <p>You cannot update a projection.</p>
<p><i>alias</i></p>	<p>A temporary name used to reference the table.</p>
<p>SET <i>set-expression</i> [,...]</p>	<p>Specifies the columns to update. Each <i>set-expression</i> in the SET clause specifies a target column and its new value as follows:</p> <pre>column-name = { expression DEFAULT }</pre> <p>where:</p> <ul style="list-style-type: none"> • <i>column-name</i> is any column that does not have primary key or foreign key referential integrity constraints. • <i>expression</i> specifies a value to assign to the column. The expression can use the current values of this and other table columns. For example: <pre>UPDATE T1 SET C1 = C1+1</pre> <p>UPDATE only modifies the columns specified by the SET clause. Unspecified columns remain unchanged.</p>
<p>FROM <i>from-list</i></p>	<p>A list of table expressions, allowing columns from other tables to appear in the WHERE condition and the UPDATE expressions. This is similar to the list of tables that can be specified in the FROM Clause of a SELECT command.</p> <p>Important: <i>from-list</i> must not include the target table.</p>

Privileges

Table owner or user with GRANT OPTION is grantor.

- UPDATE privilege on table
- USAGE privilege on schema that contains the table
- SELECT privilege on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause

Subqueries and Joins

UPDATE supports subqueries and joins, which is useful for updating values in a table based on values that are stored in other tables. For details, see [Subqueries in UPDATE and DELETE Statements](#) in Analyzing Data.

Restrictions

- The table you specify in the UPDATE list cannot also appear in the FROM list (no self joins); for example, the following UPDATE statement is not allowed:

```
=> BEGIN;
=> UPDATE result_table
   SET address='new' || r2.address
   FROM result_table r2
   WHERE r2.cust_id = result_table.cust_id + 10;
ERROR:  Self joins in UPDATE statements are not allowed
DETAIL:  Target relation result_table also appears in the FROM list
```

- If the joins specified in the WHERE predicate produce more than one copy of the row in the table to be updated, the new value of the row in the table is chosen arbitrarily.
- If any primary key, unique key, or check constraints are enabled for automatic enforcement, Vertica enforces those constraints when you insert values into a table. If a violation occurs, Vertica rolls back the SQL statement and returns an error. This behavior occurs for INSERT, UPDATE, COPY, and MERGE SQL statements. Note that automatic constraint enforcement requires that you have the SELECT privilege on the table containing the constraint.

Examples

In the fact table, modify the price column value for all rows where the cost column value is greater than 100:

```
=> UPDATE fact SET price = price - cost * 80 WHERE cost > 100;
```

In the retail.customer table, set the state column to NH when the CID column value is greater than 100:

```
=> UPDATE retail.customer SET state = 'NH' WHERE CID > 100;
```

To use table aliases in UPDATE queries, consider the following two tables:

```
=> SELECT * FROM result_table;
cust_id |      address
-----+-----
      20 | Lincoln Street
      30 | Beach Avenue
      30 | Booth Hill Road
      40 | Mt. Vernon Street
      50 | Hillside Avenue
(5 rows)
=> SELECT * FROM new_addresses;
new_cust_id | new_address
-----+-----
      20 | Infinite Loop
      30 | Loop Infinite
      60 | New Addresses
(3 rows)
```

The following query and subquery use table aliases to update the address column in result_table (alias r) with the new address from the corresponding column in the new_addresses table (alias n):

```
=> UPDATE result_table r
   SET address=n.new_address
   FROM new_addresses n
  WHERE r.cust_id = n.new_cust_id;
```

result_table shows the address field updates made for customer IDs 20 and 30:

```
=> SELECT * FROM result_table ORDER BY cust_id;
cust_id |      address
-----+-----
      20 | Infinite Loop
      30 | Loop Infinite
      30 | Loop Infinite
      40 | Mt. Vernon Street
      50 | Hillside Avenue
```

(5 rows)

Vertica System Tables

Vertica provides system tables that let you monitor your database. Query these tables the same way you perform query operations on base or temporary tables—by using SELECT statements.

For more information, see the following sections in the Administrator's Guide:

- [Using System Tables](#)
- [Monitoring Vertica](#)

V_CATALOG Schema

The system tables in this section reside in the `v_catalog` schema. These tables provide information (metadata) about the objects in a database; for example, tables, constraints, users, projections, and so on.

ACCESS_POLICY

Provides information about access policies associated with specific tables.

Column Name	Data Type	Description
ACCESS_POLICY_OID	INTEGER	A unique identifier for the access policy.
TABLE_NAME	VARCHAR	Name of the table with which the access policy is assigned.
IS_POLICY_ENABLED	BOOLEAN	Indicates whether or not you have enabled the access policy.
POLICY_TYPE	VARCHAR	The type of access policy assigned to the table, valid values are: <ul style="list-style-type: none">• Column Policy• Row Policy
EXPRESSION	VARCHAR	The expression used when creating the access policy.
TRIGGERED_BY	VARCHAR	The SQL statement that triggers the access policy.
COLUMN_NAME	VARCHAR	The column to which the access policy is assigned.

ALL_TABLES

Provides summary information about tables in a Vertica database.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema that contains the table.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the table.
TABLE_NAME	VARCHAR	The table name.
TABLE_TYPE	VARCHAR	The type of table, which can be one of the following: <ul style="list-style-type: none"> • TABLE • SYSTEM TABLE • VIEW • GLOBAL TEMPORARY • LOCAL TEMPORARY
REMARKS	VARCHAR	A brief comment about the table. You define this field by using the COMMENT ON TABLE and COMMENT ON VIEW commands.

Example

```

onenode=> SELECT DISTINCT table_name, table_type FROM all_tables
           WHERE table_name ILIKE 't%';
  table_name | table_type
-----+-----
 types      | SYSTEM TABLE
 trades     | TABLE
 tuple_mover_operations | SYSTEM TABLE
 tables     | SYSTEM TABLE
 tuning_recommendations | SYSTEM TABLE
 testid     | TABLE
 table_constraints | SYSTEM TABLE
 transactions | SYSTEM TABLE
(8 rows)
onenode=> SELECT table_name, table_type FROM all_tables
           WHERE table_name ILIKE 'my%';
  table_name | table_type
-----+-----
 mystocks   | VIEW
(1 row)
=> SELECT * FROM all_tables LIMIT 4;
-[ RECORD 1 ]-----
 schema_name | v_catalog

```

```

table_id      | 10206
table_name    | all_tables
table_type    | SYSTEM TABLE
remarks       | A complete listing of all tables and views
-[ RECORD 2 ]-----
schema_name   | v_catalog
table_id      | 10000
table_name    | columns
table_type    | SYSTEM TABLE
remarks       | Table column information
-[ RECORD 3 ]-----
schema_name   | v_catalog
table_id      | 10054
table_name    | comments
table_type    | SYSTEM TABLE
remarks       | User comments on catalog objects
-[ RECORD 4 ]-----
schema_name   | v_catalog
table_id      | 10134
table_name    | constraint_columns
table_type    | SYSTEM TABLE
remarks       | Table column constraint information

```

CLIENT_AUTH

Provides information about the client authentication methods that you created.

Column Name	Data Type	Description
AUTH_OID	INTEGER	Unique identifier for the authentication method.
AUTH_NAME	VARCHAR	Name that you gave to the authentication method.
IS_AUTH_ENABLED	BOOLEAN	Indicates whether you enabled the authentication method.
AUTH_HOST_TYPE	VARCHAR	The authentication host type, one of the following: <ul style="list-style-type: none"> LOCAL HOST HOSTSSL HOSTNOSSL
AUTH_HOST_ADDRESS	VARCHAR	If AUTH_HOST_TYPE is HOST, AUTH_HOST_ADDRESS is the IP address (or address range) of the remote host.

Column Name	Data Type	Description
AUTH_METHOD	VARCHAR	Authentication method to be used. Valid values: <ul style="list-style-type: none"> • IDENT • GSS • HASH • LDAP • REJECT • TLS • TRUST
AUTH_PARAMETERS	VARCHAR	The parameter names and values assigned to the authentication method.

Examples

This example shows how to get information about each client authentication method that you created:

```
=> SELECT auth_name, is_auth_enabled, auth_host_type, auth_host_address,
        auth_method FROM CLIENT_AUTH;
```

auth_name	is_auth_enabled	auth_host_type	auth_host_address	auth_method
v_ident	True	LOCAL		IDENT
v_gss	True	HOST	0.0.0.0/0	GSS
v_trust	False	LOCAL		TRUST
v_ldap	True	HOST	10.19.133.123/	LDAP
RejectNoSSL	True	HOSTNOSSL	0.0.0.0/0	REJECT
RejectWithSSL	True	HOSTSSL	0.0.0.0/0	REJECT
v_hash	False	LOCAL		HASH
v_tls	True	HOSTSSL	1.1.1.1/0	TLS
v_trust	True	HOSTSSL	2001:db8:ab:123/128	TLS

(9 rows)

CLIENT_AUTH_PARAMS

Provides information about client authentication methods that have parameter values assigned.

Column Name	Data Type	Description
AUTH_OID	INTEGER	A unique identifier for the authentication method.
AUTH_NAME	VARCHAR	Name that you defined for the authentication method.
AUTH_PARAMETER_NAME	VARCHAR	Parameter name required by the authentication method. Some examples are: <ul style="list-style-type: none"> • system_users • binddn_prefix • host
AUTH_PARAMETER_VALUE	VARCHAR	Value of the specified parameter.

Examples

This example shows how to retrieve parameter names and values for all authentication methods that you created. The authentication methods that have parameters are:

- v_ident
- v_ldap
- v_ldap1

```
=> SELECT * FROM CLIENT_AUTH_PARAMS;
  auth_oid | auth_name | auth_parameter_name | auth_parameter_value
-----+-----+-----+-----
 45035996273741304 | v_ident | system_users | root
 45035996273741332 | v_gss | | 
 45035996273741350 | v_password | | 
 45035996273741368 | v_trust | | 
 45035996273741388 | v_ldap | host | ldap://172.16.65.177
 45035996273741388 | v_ldap | binddn_prefix | cn=
 45035996273741388 | v_ldap | binddn_suffix | ,dc=qa_domain,dc=com
 45035996273741406 | RejectNoSSL | | 
```

```

45035996273741424 | RejectWithSSL | | |
45035996273741450 | v_md5 | | |
45035996273904044 | l_tls | | |
45035996273906566 | v_hash | | |
45035996273910432 | v_ldap1 | host | ldap://172.16.65.177
45035996273910432 | v_ldap1 | basedn | dc=qa_domain,dc=com
45035996273910432 | v_ldap1 | binddn | cn=Manager,dc=qa_domain,dc=com
45035996273910432 | v_ldap1 | bind_password | secret
45035996273910432 | v_ldap1 | search_attribute | cn
(17 rows)

```

CLUSTER_LAYOUT

Shows the relative position of the actual arrangement of the nodes participating in the cluster and the fault groups that affect them. Ephemeral nodes are not shown in the cluster layout ring because they hold no resident data.

Column Name	Data Type	Description
CLUSTER_POSITION	INTEGER	Position of the node in the cluster ring, counting forward from 0. Note: An output value of 0 has no special meaning other than there are no nodes in position before the node assigned 0.
NODE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the node.
NODE_NAME	VARCHAR	The name of the node in the cluster ring. Only permanent nodes participating in database activity appear in the cluster layout. Ephemeral nodes are not shown in the output.
FAULT_GROUP_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the fault group. Note: This value matches the FAULT_GROUP.MEMBER_ID value, but only if this node is in a fault group; otherwise the value is NULL.
FAULT_GROUP_NAME	VARCHAR	The name of the fault group for the node.
FAULT_GROUP_TIER	INTEGER	The node's depth in the fault group tree hierarchy. For

Column Name	Data Type	Description
		<p>example is the node:</p> <ul style="list-style-type: none"> • Is not in a fault group, output is null • Is in the top level fault group, output is 0 • Is in a fault group's child, output is 1 • Is a fault group's grandchild, output is 2

See Also

[Large Cluster](#) in the Administrator's Guide

COLUMNS

Provides table column information.

Column Name	Data Type	Description
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the table.
TABLE_SCHEMA	VARCHAR	Schema name for which information is listed in the database.
TABLE_NAME	VARCHAR	Table name for which information is listed in the database.
IS_SYSTEM_TABLE	BOOLEAN	Specifies whether the table is a system table.
COLUMN_ID	VARCHAR	A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.
COLUMN_NAME	VARCHAR	The column name for which information is listed in the database.
DATA_TYPE	VARCHAR	Column's data type, for example VARCHAR(16), INT, or FLOAT.
DATA_TYPE_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog,

Column Name	Data Type	Description
		which identifies the data type.
DATA_TYPE_LENGTH	INTEGER	Maximum allowable length of the data type.
CHARACTER_MAXIMUM_LENGTH	VARCHAR	Maximum allowable length of the column.
NUMERIC_PRECISION	INTEGER	Number of significant decimal digits.
NUMERIC_SCALE	INTEGER	Number of fractional digits.
DATETIME_PRECISION	INTEGER	For <code>TIMESTAMP</code> data type, returns the declared precision; returns <code>NULL</code> if no precision was declared.
INTERVAL_PRECISION	INTEGER	Number of fractional digits retained in the seconds field.
ORDINAL_POSITION	INTEGER	Column position relative to other columns in the table.
IS_NULLABLE	BOOLEAN	Specifies whether the column can contain <code>NULL</code> values.
COLUMN_DEFAULT	VARCHAR	Expression set on a column with the constraint DEFAULT .
IS_IDENTITY	BOOLEAN	Specifies whether the column is an identity column. See Column-Constraint .

Examples

Retrieve table and column information from the `COLUMNS` table:

```
=> SELECT table_schema, table_name, column_name, data_type, is_nullable
      FROM columns WHERE table_schema = 'store'
      AND data_type = 'Date';
table_schema | table_name      | column_name      | data_type | is_nullable
-----+-----+-----+-----+-----
store        | store_dimension | first_open_date  | Date      | f
store        | store_dimension | last_remodel_date | Date      | f
store        | store_orders_fact | date_ordered     | Date      | f
store        | store_orders_fact | date_shipped     | Date      | f
store        | store_orders_fact | expected_delivery_date | Date      | f
store        | store_orders_fact | date_delivered   | Date      | f
6 rows)
```

`DATETIME_PRECISION` is `NULL` because the table definition declares no precision:

```
=> CREATE TABLE c (c TIMESTAMP);
CREATE TABLE
=> SELECT table_name, column_name, datetime_precision FROM columns
  WHERE table_name = 'c';
 table_name | column_name | datetime_precision
-----+-----+-----
c          | c          |
(1 row)
```

DATETIME_PRECISION is 4 because the table definition declares precision as 4:

```
=> DROP TABLE c;
=> CREATE TABLE c (c TIMESTAMP(4));
CREATE TABLE
=> SELECT table_name, column_name, datetime_precision FROM columns
  WHERE table_name = 'c';
 table_name | column_name | datetime_precision
-----+-----+-----
c          | c          | 4
```

An identity column is a sequence available only for numeric column types. To identify what column in a table, if any, is an identity column, search the COLUMNS table to find the identity column in a table testid:

```
=> CREATE TABLE testid (c1 IDENTITY(1, 1, 1000), c2 INT);
=> \x
Expanded display is on.
=> SELECT * FROM COLUMNS WHERE is_identity='t' AND table_name='testid';
-[ RECORD 1 ]-----+-----
table_id          | 45035996273719486
table_schema      | public
table_name        | testid
is_system_table   | f
column_id         | 45035996273719486-1
column_name       | c1
data_type         | int
data_type_id      | 6
data_type_length  | 8
character_maximum_length |
numeric_precision |
numeric_scale     |
datetime_precision |
interval_precision |
ordinal_position  | 1
is_nullable       | f
column_default    |
is_identity       | t
```

Use the SEQUENCES table to get detailed information about the sequence in testid:

```
=> SELECT * FROM sequences WHERE identity_table_name='testid';
-[ RECORD 1 ]-----+-----
sequence_schema | public
sequence_name   | testid_c1_seq
owner_name      | dbadmin
```

```

identity_table_name | testid
session_cache_count | 1000
allow_cycle         | f
output_ordered      | f
increment_by        | 1
minimum             | 1
maximum             | 9223372036854775807
current_value       | 0
sequence_schema_id | 45035996273704976
sequence_id         | 45035996273719488
owner_id            | 45035996273704962
identity_table_id   | 45035996273719486

```

For more information about sequences and identity columns, see [Working with Sequence Types](#).

COMMENTS

Returns information about comments associated with objects in the database.

Column Name	Data Type	Description
COMMENT_ID	INTEGER	The comment's internal ID number
OBJECT_ID	INTEGER	The internal ID number of the object associated with the comment
OBJECT_TYPE	VARCHAR	The type of object associated with the comment. Possible values are: <ul style="list-style-type: none"> • COLUMN • CONSTRAINT • FUNCTION • LIBRARY • NODE • PROJECTION • SCHEMA • SEQUENCE • TABLE

Column Name	Data Type	Description
		<ul style="list-style-type: none"> VIEW
OBJECT_SCHEMA	VARCHAR	The schema containing the object.
OBJECT_NAME	VARCHAR	The name of the object associated with the comment.
OWNER_ID	VARCHAR	The internal ID of the owner of the object.
OWNER_NAME	VARCHAR	The object owner's name.
CREATION_TIME	TIMESTAMPTZ	When the comment was created.
LAST_MODIFIED_TIME	TIMESTAMPTZ	When the comment was last modified.
COMMENT	VARCHAR	The text of the comments.

CONSTRAINT_COLUMNS

Records information about table column constraints.

Column Name	Data Type	Description
CONSTRAINT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
TABLE_SCHEMA	VARCHAR	Name of the schema that contains this table.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the table.
TABLE_NAME	VARCHAR	Name of the table in which the column resides.
COLUMN_NAME	VARCHAR	Name of the column that is constrained. For check constraints, if more than one column is referenced, each appears as a separate row.
CONSTRAINT_NAME	VARCHAR	Constraint name for which information is listed.
CONSTRAINT_TYPE	CHAR	Indicates the constraint type. Valid Values:

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • c — check • f — foreign • n — not null • p — primary • u — unique
IS_ENABLED	BOOLEAN	Indicates if a constraint for a primary key, unique key, or check constraint is currently enabled. Can be <i>t</i> (True) or <i>f</i> (False).
REFERENCE_TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the referenced table
REFERENCE_TABLE_SCHEMA	VARCHAR	Schema name for which information is listed.
REFERENCE_TABLE_NAME	VARCHAR	References the TABLE_NAME column in the PRIMARY_KEY table.
REFERENCE_COLUMN_NAME	VARCHAR	References the COLUMN_NAME column in the PRIMARY_KEY table.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

DATABASES

Provides information about the databases in this Vertica installation.

Column Name	Data Type	Description
DATABASE_ID	INTEGER	The database's internal ID number
DATABASE_NAME	VARCHAR	The database's name

Column Name	Data Type	Description
OWNER_ID	INTEGER	The database owner's ID
OWNER_NAME	INTEGER	The database owner's name
START_TIME	TIMESTAMPTZ	The date and time the database last started
COMPLIANCE_MESSAGE	VARCHAR	Message describing the current state of the database's license compliance.
EXPORT_SUBNET	VARCHAR	The subnet (on the public network) used by the database for import/export.
LOAD_BALANCE_POLICY	VARCHAR	The current native connection load balance policy, which controls whether client connection requests are redirected to other hosts in the database. See About Native Connection Load Balancing in the Administrator's Guide.

DIRECTED_QUERIES

Returns information about directed queries.

Column Name	Data Type	Description
query_name	VARCHAR	This directed query's unique identifier, used by statements such as ACTIVATE DIRECTED QUERY .
is_active	BOOLEAN	Specifies whether the directed query is active.
vertica_version	VARCHAR	The Vertica version used when this directed query was created.
comment	VARCHAR	A user-supplied comment specified on creation of the directed query, up to 128 characters.
creation_date	TIMESTAMPTZ	Specifies when the directed query was created.
input_query	VARCHAR	The input query that is associated with this directed query. Multiple directed queries can map to the same

Column Name	Data Type	Description
		input query.
annotated_query	VARCHAR	The directed query that was saved with CREATE DIRECTED QUERY .

Privileges

Superuser

Truncated Query Results

Query results for the fields `input_query` and `annotated_query` are truncated after 8192 characters. You can get the full content of both fields in two ways:

- Use the statement [GET DIRECTED QUERY](#).
- Use [EXPORT_CATALOG](#) to export directed queries.

DUAL

DUAL is a single-column "dummy" table with one record whose value is X; for example:

```
=> SELECT * FROM DUAL;
  dummy
-----
      X
(1 row)
```

You can write the following types of queries:

```
=> SELECT 1 FROM dual;
 ?column?
-----
        1
(1 row)
=> SELECT current_timestamp, current_user FROM dual;
 ?column? | current_user
-----+-----
2010-03-08 12:57:32.065841-05 | release
(1 row)
=> CREATE TABLE t1(col1 VARCHAR(20), col2 VARCHAR(2));
=> INSERT INTO T1(SELECT 'hello' AS col1, 1 AS col2 FROM dual);
=> SELECT * FROM t1;
```

```

col1 | col2
-----+-----
hello | 1
(1 row

```

Restrictions

You cannot create projections for DUAL.

ELASTIC_CLUSTER

Returns information about cluster elasticity, such as whether [Elastic Cluster](#) is running.

Column Name	Data Type	Description
SCALING_FACTOR	INTEGER	This value is only meaningful when you enable local segments. SCALING_FACTOR influences the number of local segments on each node. Initially—before a rebalance runs—there are <i>scaling_factor</i> number of local segments per node. A large SCALING_FACTOR is good for rebalancing a potentially wide range of cluster configurations quickly. However, too large a value could lead to ROS pushback, particularly in a database with a table with a large number of partitions. See SET_SCALING_FACTOR for more details.
MAXIMUM_SKEW_PERCENT	INTEGER	This value is only meaningful when you enable local segments. MAXIMUM_SKEW_PERCENT is the maximum amount of skew a rebalance operation tolerates, which preferentially redistributes local segments; however, if after doing so the segment ranges of any two nodes differs by more than this amount, rebalance will separate and distribute storage to even the distribution.
SEGMENT_LAYOUT	VARCHAR	Current, offset=0, segment layout. New segmented projections will be created with this layout, with segments rotated by the corresponding offset.

Column Name	Data Type	Description
		Existing segmented projections will be rebalanced into an offset of this layout.
LOCAL_SEGMENT_LAYOUT	VARCHAR	Similar to SEGMENT_LAYOUT but includes details that indicate the number of local segments, their relative size and node assignment.
VERSION	INTEGER	Number that gets incremented each time the cluster topology changes (nodes added, marked ephemeral, marked permanent, etc). Useful for monitoring active and past rebalance operations.
IS_ENABLED	BOOLEAN	True if Elastic Cluster is enabled, otherwise false.
IS_LOCAL_SEGMENT_ENABLED	BOOLEAN	True if local segments are enabled, otherwise false.
IS_REBALANCE_RUNNING	BOOLEAN	True if rebalance is currently running, otherwise false.

Privileges

Superuser

See Also

- [ENABLE_ELASTIC_CLUSTER](#)
- [DISABLE_ELASTIC_CLUSTER](#)
- [Elastic Cluster](#)

EPOCHS

For all epochs, provides the date and time of the close and the corresponding epoch number of the closed epoch. This information lets you determine which time periods pertain to which epochs.

Column Name	Data Type	Description
EPOCH_CLOSE_TIME	DATETIME	The date and time of the close of the epoch.
EPOCH_NUMBER	INTEGER	The corresponding epoch number of the closed epoch.

See Also

- [Epoch Management Parameters](#)
- [Epoch Management Functions](#)

FAULT_GROUPS

View the fault groups and their hierarchy in the cluster.

Column Name	Data Type	Description
MEMBER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the fault group.
MEMBER_TYPE	VARCHAR	The type of fault group. Values can be either NODE or FAULT GROUP.
MEMBER_NAME	VARCHAR	Name associated with this fault group. Values will be the node name or the fault group name.
PARENT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the parent fault group. The parent fault group can contain: <ul style="list-style-type: none"> • Nodes • Other fault groups • Nodes and other fault groups
PARENT_TYPE	VARCHAR	The type of parent fault group, where the default/root parent is the DATABASE object. Can be one of the following objects: <ul style="list-style-type: none"> • FAULT GROUP

Column Name	Data Type	Description
		<ul style="list-style-type: none"> DATABASE
PARENT_NAME	VARCHAR	The name of the fault group that contains nodes or other fault groups or both nodes and fault groups.
IS_AUTOMATICALLY_GENERATED	BOOLEAN	If true, denotes whether Vertica Analytics Platform created fault groups for you to manage the fault tolerance of control nodes in large cluster configurations. If false, denotes that you created fault groups manually. See Fault Groups for more information

Examples

Show the current hierarchy of fault groups in the cluster:

```
vmartdb=> SELECT member_type, member_name, parent_type, CASE
            WHEN parent_type = 'DATABASE' THEN ''
            ELSE parent_name END FROM fault_groups
            ORDER BY member_name;
member_type | member_name          | parent_type | parent_name
-----+-----+-----+-----
NODE        | v_vmart_node0001    | FAULT GROUP | two
NODE        | v_vmart_node0002    | FAULT GROUP | two
NODE        | v_vmart_node0003    | FAULT GROUP | three
FAULT GROUP | one                  | DATABASE    |
FAULT GROUP | three                | DATABASE    |
FAULT GROUP | two                  | FAULT GROUP | one
```

View the distribution of the segment layout:

```
vmartdb=> SELECT segment_layout from elastic_cluster;
            segment_layout
-----+-----+-----+-----
v_vmart_node0001[33.3%] v_vmart_node0003[33.3%] v_vmart_node0004[33.3%]
(1 row)
```

See Also

- [High Availability With Fault Groups](#) in Vertica Concepts
- [Fault Groups](#) in the Administrator's Guide

FOREIGN_KEYS

Provides foreign key information.

Column Name	Data Type	Description
CONSTRAINT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
CONSTRAINT_NAME	VARCHAR	The constraint name for which information is listed.
COLUMN_NAME	VARCHAR	The name of the column that is constrained.
ORDINAL_POSITION	VARCHAR	The position of the column within the key. The numbering of columns starts at 1.
TABLE_NAME	VARCHAR	The table name for which information is listed.
REFERENCE_TABLE_NAME	VARCHAR	References the TABLE_NAME column in the PRIMARY_KEY table.
CONSTRAINT_TYPE	VARCHAR	The constraint type, f, for foreign key.
REFERENCE_COLUMN_NAME	VARCHAR	References the COLUMN_NAME column in the PRIMARY_KEY table.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
REFERENCE_TABLE_SCHEMA	VARCHAR	References the TABLE_SCHEMA column in the PRIMARY_KEY table.

Example

```
mydb=> SELECT
    constraint_name,
    table_name,
    ordinal_position,
    reference_table_name
FROM foreign_keys ORDER BY 3;
constraint_name | table_name | ordinal_position | reference_table_name
-----+-----+-----+-----
fk_store_sales_date | store_sales_fact | 1 | date_dimension
fk_online_sales_saledate | online_sales_fact | 1 | date_dimension
fk_store_orders_product | store_orders_fact | 1 | product_dimension
```

```

fk_inventory_date      | inventory_fact      |          1 | date_dimension
fk_inventory_product  | inventory_fact      |          2 | product_dimension
fk_store_sales_product | store_sales_fact    |          2 | product_dimension
fk_online_sales_shipdate | online_sales_fact  |          2 | date_dimension
fk_store_orders_product | store_orders_fact  |          2 | product_dimension
fk_inventory_product  | inventory_fact      |          3 | product_dimension
fk_store_sales_product | store_sales_fact    |          3 | product_dimension
fk_online_sales_product | online_sales_fact  |          3 | product_dimension
fk_store_orders_store | store_orders_fact  |          3 | store_dimension
fk_online_sales_product | online_sales_fact  |          4 | product_dimension
fk_inventory_warehouse | inventory_fact      |          4 | warehouse_dimension
fk_store_orders_vendor | store_orders_fact  |          4 | vendor_dimension
fk_store_sales_store  | store_sales_fact    |          4 | store_dimension
fk_store_orders_employee | store_orders_fact |          5 | employee_dimension
fk_store_sales_promotion | store_sales_fact  |          5 | promotion_dimension
fk_online_sales_customer | online_sales_fact |          5 | customer_dimension
fk_store_sales_customer | store_sales_fact    |          6 | customer_dimension
fk_online_sales_cc     | online_sales_fact  |          6 | call_center_dimension
fk_store_sales_employee | store_sales_fact    |          7 | employee_dimension
fk_online_sales_op     | online_sales_fact  |          7 | online_page_dimension
fk_online_sales_shipping | online_sales_fact  |          8 | shipping_dimension
fk_online_sales_warehouse | online_sales_fact |          9 | warehouse_dimension
fk_online_sales_promotion | online_sales_fact |         10 | promotion_dimension
(26 rows)

```

GRANTS

Provides information about privileges granted on various objects, the granting user, and grantee user. The order of columns in the table corresponds to the order in which they appear in the GRANT command. The GRANTS table does not retain the role grantor.

Column Name	Data Type	Description
GRANTEE	VARCHAR	The user being granted permission.
GRANTEE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user granted permissions.
GRANT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the
GRANTOR	VARCHAR	The user granting the permission.
GRANTOR_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user who performed the grant operation.
OBJECT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog,

Column Name	Data Type	Description
		which identifies the object granted.
OBJECT_NAME	VARCHAR	The name of the object that is being granted privileges. Note that for schema privileges, the schemaname appears in the OBJECT_NAME column instead of the OBJECT_SCHEMA column.
OBJECT_SCHEMA	VARCHAR	The name of the schema that is being granted privileges.
OBJECT_TYPE	VARCHAR	The object type on which the grant was applied; for example, ROLE, SCHEMA, DATABASE, RESOURCEPOOL. Output from this column is useful in cases where a schema, resource pool, or user share the same name.
PRIVILEGES_DESCRIPTION	VARCHAR	A readable description of the privileges being granted; for example INSERT, SELECT. An asterisk in PRIVILEGES_DESCRIPTION output indicates a privilege WITH GRANT OPTION.

Notes

The vsql commands \dp and \z both include the schema name in the output. For example:

```
=> \dp
      Access privileges for database "vmartdb"
  Grantee | Grantor | Privileges | Schema |      Name
-----+-----+-----+-----+-----
          | dbadmin | USAGE     |        | public
          | dbadmin | USAGE     |        | v_internal
          | dbadmin | USAGE     |        | v_catalog
          | dbadmin | USAGE     |        | v_monitor
          | dbadmin | USAGE     |        | v_internal
          | dbadmin | USAGE     |        | v_catalog
          | dbadmin | USAGE     |        | v_monitor
          | dbadmin | USAGE     |        | v_internal
          | dbadmin | USAGE     |        | designer_system
(9 rows)
```

The vsql command \dp *.tablename displays table names in all schemas. This command lets you distinguish grants for same-named tables in different schemas:

```
=> \dp *.events
Access privileges for database "dbadmin"
Grantor | Grantor | Privileges | Schema | Name
-----+-----+-----+-----+-----
user2   | dbadmin | INSERT, SELECT, UPDATE, DELETE, REFERENCES | schema1 | events
user1   | dbadmin | SELECT | schema1 | events
user2   | dbadmin | INSERT, SELECT, UPDATE, DELETE, REFERENCES | schema2 | events
user1   | dbadmin | INSERT, SELECT | schema2 | events
(4 rows)
```

The vsql command `\dp schemaname.*` displays all tables in the named schema:

```
=> \dp schema1.*
Access privileges for database "dbadmin"
Grantor | Grantor | Privileges | Schema | Name
-----+-----+-----+-----+-----
user2   | dbadmin | INSERT, SELECT, UPDATE, DELETE, REFERENCES | schema1 | events
user1   | dbadmin | SELECT | schema1 | events
(2 rows)
```

Examples

This example shows CREATE and USAGE privileges granted to Bob in the fictitious apps database:

```
=> SELECT grantor, privileges_description, object_schema, object_name, grantee
FROM grants;
grantor | privileges_description | object_schema | object_name | grantee
-----+-----+-----+-----+-----
dbadmin | USAGE | | general | Bob
dbadmin | CREATE | | schema2 | Bob
(2 rows)
```

This next query looks for privileges granted to a particular set of grantees. The asterisk in `privileges_description` column for User1 means that user has WITH GRANT OPTION privileges.

```
=> SELECT grantor, privileges_description, object_schema, object_name, grantee
FROM grants WHERE grantee ILIKE 'User%';
grantor | privileges_description | object_schema | object_name | grantee
-----+-----+-----+-----+-----
release | USAGE | | general | User1
release | USAGE | | general | User2
release | USAGE | | general | User3
release | USAGE | | s1 | User1
release | USAGE | | s1 | User2
release | USAGE | | s1 | User3
User1 | INSERT*, SELECT*, UPDATE* | s1 | t1 | User1
(7 rows)
```

In the following example, `online_sales` is the schema that first gets privileges, and then inside that schema the anchor table gets SELECT privileges:

```
=> SELECT grantee, grantor, privileges_description, object_schema, object_name
      FROM grants WHERE grantee='u1' ORDER BY object_name;
```

grantee	grantor	privileges_description	object_schema	object_name
u1	dbadmin	CREATE		online_sales
u1	dbadmin	SELECT	online_sales	online_sales_fact

The following statement shows all grants for user Bob:

```
-> SELECT * FROM GRANTS WHERE grantee = 'Bob';
-[ RECORD 1 ]-----+-----
grant_id          | 45035996273749244
grantor_id        | 45035996273704962
grantor           | dbadmin
privileges_description | USAGE
object_schema     |
object_name       | general
object_id         | 45035996273718666
object_type       | RESOURCEPOOL
grantee_id        | 45035996273749242
grantee           | Bob
-[ RECORD 2 ]-----+-----
grant_id          | 45035996273749598
grantor_id        | 45035996273704962
grantor           | dbadmin
privileges_description |
object_schema     |
object_name       | dbadmin
object_id         | 45035996273704968
object_type       | ROLE
grantee_id        | 45035996273749242
grantee           | Bob
-[ RECORD 3 ]-----+-----
grant_id          | 45035996273749716
grantor_id        | 45035996273704962
grantor           | dbadmin
privileges_description |
object_schema     |
object_name       | dbadmin
object_id         | 45035996273704968
object_type       | ROLE
grantee_id        | 45035996273749242
grantee           | Bob
-[ RECORD 4 ]-----+-----
grant_id          | 45035996273755986
grantor_id        | 45035996273704962
grantor           | dbadmin
privileges_description |
object_schema     |
object_name       | pseudosuperuser
object_id         | 45035996273704970
object_type       | ROLE
grantee_id        | 45035996273749242
grantee           | Bob
-[ RECORD 5 ]-----+-----
grant_id          | 45035996273756986
grantor_id        | 45035996273704962
```

```

grantor          | dbadmin
privileges_description | CREATE, CREATE TEMP
object_schema   |
object_name     | mcdb
object_id       | 45035996273704974
object_type     | DATABASE
grantee_id      | 45035996273749242
grantee         | Bob
-[ RECORD 5 ]-----+-----
...

```

See Also

- [HAS_ROLE](#)
- [ROLES](#)
- [USERS](#)
- [Managing Users and Privileges](#)

HCATALOG_COLUMNS

Describes the columns of all tables available through the HCatalog Connector. Each row in this table corresponds to to a column in a table accessible through the HCatalog Connector. See [Using the HCatalog Connector](#) in Integrating with Apache Hadoop for more information.

Column Name	Data Type	Description
TABLE_SCHEMA	VARCHAR (128)	The name of the Vertica Analytics Platform schema that contains the table containing this column
HCATALOG_SCHEMA	VARCHAR (128)	The name of the Hive schema or database that contains the table containing this column
TABLE_NAME	VARCHAR (128)	The name of the table that contains the column
IS_PARTITION_COLUMN	BOOLEAN	Whether the table is partitioned on this column
COLUMN_NAME	VARCHAR (128)	The name of the column

Column Name	Data Type	Description
HCATALOG_DATA_TYPE	VARCHAR (128)	The Hive data type of this column
DATA_TYPE	VARCHAR (128)	The Vertica Analytics Platform data type of this column
DATA_TYPE_ID	INTEGER	Numeric ID of the column's Vertica Analytics Platform data type
DATA_TYPE_LENGTH	INTEGER	The number of bytes used to store this data type
CHARACTER_MAXIMUM_LENGTH	INTEGER	For string data types, the maximum number of characters it can hold
NUMERIC_PRECISION	INTEGER	For numeric types, the precision of the values in the column
NUMERIC_SCALE	INTEGER	For numeric data types, the scale of the values in the column
DATETIME_PRECISION	INTEGER	For datetime data types, the precision of the values in the column
INTERVAL_PRECISION	INTEGER	For interval data types, the precision of the values in the column
ORDINAL_POSITION	INTEGER	The position of the column within the table

Privileges

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

Notes

If you are using WebHCat instead of HiveServer2, querying this table results in one web service call to the WebHCat server for each table in each HCatalog schema. If you need to perform multiple queries on this table in a short period of time, consider creating a copy of the table

using a CREATE TABLE AS statement to improve performance. The copy does not reflect any changes made to the schema of the Hive tables after it was created, but it is much faster to query.

Example

The following example demonstrates finding the column information for a specific table:

```
=> SELECT * FROM HCATALOG_COLUMNS WHERE table_name = 'hcatalogtypes'
-> ORDER BY ordinal_position;
-[ RECORD 1 ]-----+-----
table_schema      | hcat
hcatalog_schema   | default
table_name        | hcatalogtypes
is_partition_column | f
column_name       | intcol
hcatalog_data_type | int
data_type         | int
data_type_id      | 6
data_type_length  | 8
character_maximum_length |
numeric_precision |
numeric_scale     |
datetime_precision |
interval_precision |
ordinal_position  | 1
-[ RECORD 2 ]-----+-----
table_schema      | hcat
hcatalog_schema   | default
table_name        | hcatalogtypes
is_partition_column | f
column_name       | floatcol
hcatalog_data_type | float
data_type         | float
data_type_id      | 7
data_type_length  | 8
character_maximum_length |
numeric_precision |
numeric_scale     |
datetime_precision |
interval_precision |
ordinal_position  | 2
-[ RECORD 3 ]-----+-----
table_schema      | hcat
hcatalog_schema   | default
table_name        | hcatalogtypes
is_partition_column | f
column_name       | doublecol
hcatalog_data_type | double
data_type         | float
data_type_id      | 7
data_type_length  | 8
character_maximum_length |
numeric_precision |
numeric_scale     |
datetime_precision |
```

```

interval_precision |
ordinal_position   | 3
-[ RECORD 4 ]-----+-----
table_schema       | hcat
hcatalog_schema    | default
table_name         | hcatalogtypes
is_partition_column | f
column_name        | charcol
hcatalog_data_type | string
data_type          | varchar(65000)
data_type_id       | 9
data_type_length   | 65000
character_maximum_length | 65000
numeric_precision |
numeric_scale      |
datetime_precision |
interval_precision |
ordinal_position   | 4
-[ RECORD 5 ]-----+-----
table_schema       | hcat
hcatalog_schema    | default
table_name         | hcatalogtypes
is_partition_column | f
column_name        | varcharcol
hcatalog_data_type | string
data_type          | varchar(65000)
data_type_id       | 9
data_type_length   | 65000
character_maximum_length | 65000
numeric_precision |
numeric_scale      |
datetime_precision |
interval_precision |
ordinal_position   | 5
-[ RECORD 6 ]-----+-----
table_schema       | hcat
hcatalog_schema    | default
table_name         | hcatalogtypes
is_partition_column | f
column_name        | boolcol
hcatalog_data_type | boolean
data_type          | boolean
data_type_id       | 5
data_type_length   | 1
character_maximum_length |
numeric_precision |
numeric_scale      |
datetime_precision |
interval_precision |
ordinal_position   | 6
-[ RECORD 7 ]-----+-----
table_schema       | hcat
hcatalog_schema    | default
table_name         | hcatalogtypes
is_partition_column | f
column_name        | timestampcol
hcatalog_data_type | string
data_type          | varchar(65000)
data_type_id       | 9
data_type_length   | 65000

```

```
character_maximum_length | 65000
numeric_precision        |
numeric_scale            |
datetime_precision       |
interval_precision       |
ordinal_position         | 7
-[ RECORD 8 ]-----+-----
table_schema             | hcat
hcatalog_schema          | default
table_name               | hcatalogtypes
is_partition_column      | f
column_name              | varbincol
hcatalog_data_type       | binary
data_type                | varbinary(65000)
data_type_id             | 17
data_type_length         | 65000
character_maximum_length | 65000
numeric_precision        |
numeric_scale            |
datetime_precision       |
interval_precision       |
ordinal_position         | 8
-[ RECORD 9 ]-----+-----
table_schema             | hcat
hcatalog_schema          | default
table_name               | hcatalogtypes
is_partition_column      | f
column_name              | bincol
hcatalog_data_type       | binary
data_type                | varbinary(65000)
data_type_id             | 17
data_type_length         | 65000
character_maximum_length | 65000
numeric_precision        |
numeric_scale            |
datetime_precision       |
interval_precision       |
ordinal_position         | 9
```

See Also

- [HCATALOG_SCHEMATA](#)
- [HCATALOG_TABLES](#)
- [HCATALOG_TABLE_LIST](#)

HCATALOG_SCHEMATA

Lists all of the schemas defined using the HCatalog Connector. See [Using the HCatalog Connector](#) in Integrating with Apache Hadoop.

Unlike other HCatalog Connector-related system tables, this table makes no calls to Hive, so querying incurs very little overhead.

Column Name	Data Type	Description
SCHEMA_ID	INTEGER	The Vertica Analytics Platform ID number for the schema
SCHEMA_NAME	VARCHAR(128)	The name of the schema defined in the Vertica Analytics Platform catalog
SCHEMA_OWNER_ID	INTEGER	The ID number of the user who owns the Vertica Analytics Platform schema
SCHEMA_OWNER	VARCHAR(128)	The username of the Vertica Analytics Platform schema's owner
CREATE_TIME	TIMESTAMPTZ	The date and time the schema as created
HOSTNAME	VARCHAR(128)	The host name or IP address of the database server that holds the Hive metadata
PORT	INTEGER	The port number on which the metastore database listens for connections
HIVESERVER2_HOSTNAME	VARCHAR(128)	The host name or IP address of the HiveServer2 server for the Hive database
WEBSERVICE_HOSTNAME	VARCHAR(128)	The host name or IP address of the WebHCat server for the Hive database, if used
WEBSERVICE_PORT	INTEGER	The port number on which the WebHCat server listens for connections
WEBHDFS_ADDRESS	VARCHAR (128)	The host and port ("host:port") for the WebHDFS service, used for reading ORC and Parquet files
HCATALOG_SCHEMA_NAME	VARCHAR(128)	The name of the schema or database in Hive to which the Vertica Analytics Platform schema is mapped/
HCATALOG_USER_NAME	VARCHAR(128)	The username the HCatalog Connector uses to authenticate itself to the Hive database.
METASTORE_DB_NAME	VARCHAR(128)	The name of the database containing the metadata about the Hive database.

Privileges

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

See Also

- [HCATALOG_COLUMNS](#)
- [HCATALOG_TABLE_LIST](#)
- [HCATALOG_TABLES](#)

HCATALOG_TABLES

Returns a detailed list of all tables made available through the HCatalog Connector. See [Using the HCatalog Connector](#) in Integrating with Apache Hadoop.

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	ID number of the schema
TABLE_SCHEMA	VARCHAR(128)	The name of the Vertica Analytics Platform schema through which the table is available
HCATALOG_SCHEMA	VARCHAR(128)	The name of the Hive schema or database that contains the table
TABLE_NAME	VARCHAR(128)	The name of the table
HCATALOG_USER_NAME	VARCHAR(128)	The name of the HCatalog user whose credentials are used to access the table's data
MIN_FILE_SIZE_BYTES	INTEGER	The file size of the table's smallest data file, if using WebHCat; null if using HiveServer2
TOTAL_NUMBER_FILES	INTEGER	The number of files used to store this table's data in HDFS
LOCATION	VARCHAR(8192)	The URI for the directory containing this table's data,

Column Name	Data Type	Description
		normally an HDFS URI
LAST_UPDATE_TIME	TIMESTAMPTZ	The last time data in this table was updated, if using WebHCat; null if using HiveServer2
OUTPUT_FORMAT	VARCHAR(128)	The Hive SerDe class used to output data from this table
LAST_ACCESS_TIME	TIMESTAMPTZ	The last time data in this table was accessed, if using WebHCat; null if using HiveServer2
MAX_FILE_SIZE_BYTES	INTEGER	The size of the largest data file for this table, if using WebHCat; null if using HiveServer2
IS_PARTITIONED	BOOLEAN	Whether this table is partitioned
PARTITION_EXPRESSION	VARCHAR(128)	The expression used to partition this table
TABLE_OWNER	VARCHAR(128)	The Hive user that owns this table in the Hive database, if using WebHCat; null if using HiveServer2
INPUT_FORMAT	VARCHAR(128)	The SerDe class used to read the data from this table
TOTAL_FILE_SIZE_BYTES	INTEGER	Total number of bytes used by all of this table's data files
HCATALOG_GROUP	VARCHAR(128)	The permission group assigned to this table, if using WebHCat; null if using HiveServer2
PERMISSION	VARCHAR(128)	The Unix file permissions for this group, as shown by the <code>ls -l</code> command, if using WebHCat; null if using HiveServer2

Privileges

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

See Also

- [HCATALOG_SCHEMATA](#)
- [HCATALOG_COLUMNS](#)
- [HCATALOG_TABLE_LIST](#)

HCATALOG_TABLE_LIST

A concise list of all tables contained in all Hive schemas and databases available through the HCatalog Connector. See [Using the HCatalog Connector](#) in Integrating with Apache Hadoop.

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	Internal ID number for the schema containing the table
TABLE_SCHEMA	VARCHAR (128)	Name of the Vertica Analytics Platform schema through which the table is available
HCATALOG_SCHEMA	VARCHAR (128)	Name of the Hive schema or database containing the table
TABLE_NAME	VARCHAR (128)	The name of the table
HCATALOG_USER_NAME	VARCHAR (128)	Name of Hive user used to access the table

Privileges

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

Notes

- Querying this table results in one call to HiveServer2 for each Hive schema defined using the HCatalog Connector. This means that the query usually takes longer than querying other system tables.
- Querying this table is faster than querying HCATALOG_TABLES. Querying HCATALOG_TABLE_LIST only makes one HiveServer2 call per HCatalog schema versus one call per table for HCATALOG_TABLES.

Example

The following example demonstrates defining a new HCatalog schema then querying HCATALOG_TABLE_LIST. Note that one table defined in a different HCatalog schema also appears. HCATALOG_TABLE_LIST lists all of the tables available in any of the HCatalog schemas:

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcatost'
-> HCATALOG_SCHEMA='default' HCATALOG_DB='default' HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> \x
Expanded display is on.
=> SELECT * FROM v_catalog.hcatalog_table_list;
-[ RECORD 1 ]-----+-----
table_schema_id | 45035996273748980
table_schema    | hcat
hcatalog_schema | default
table_name      | weblogs
hcatalog_user_name | hcatuser
-[ RECORD 2 ]-----+-----
table_schema_id | 45035996273748980
table_schema    | hcat
hcatalog_schema | default
table_name      | tweets
hcatalog_user_name | hcatuser
-[ RECORD 3 ]-----+-----
table_schema_id | 45035996273748980
table_schema    | hcat
hcatalog_schema | default
table_name      | messages
hcatalog_user_name | hcatuser
-[ RECORD 4 ]-----+-----
table_schema_id | 45035996273864948
table_schema    | hiveschema
hcatalog_schema | default
table_name      | weblogs
hcatalog_user_name | hcatuser
```

See Also

- [HCATALOG_COLUMNS](#)
- [HCATALOG_SCHEMATA](#)
- [HCATALOG_TABLES](#)

KEYWORDS

Identifies Vertica reserved and non-reserved keywords.

Column Name	Data Type	Description
KEYWORD	VARCHAR	Vertica reserved or non-reserved keyword.
RESERVED	VARCHAR	Indicates whether a keyword is reserved or non-reserved: <ul style="list-style-type: none">• R: reserved• N: non-reserved

See Also

[Keywords](#)

LARGE_CLUSTER_CONFIGURATION_STATUS

Shows the current cluster nodes and control node (spread hosts) designations in the Catalog so you can see if they match.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node in the cluster.
SPREAD_HOST_NAME	VARCHAR	The host name of the control node (the host that manages control message responsibilities)

Column Name	Data Type	Description
CONTROL_NODE_NAME	VARCHAR	The name of the control node

See Also

[Large Cluster](#) in the Administrator's Guide

LICENSE_AUDITS

Lists the results of Vertica's license automatic compliance audits. See [How Vertica Calculates Database Size](#) in the Administrator's Guide.

Column Name	Data Type	Description
DATABASE_SIZE_BYTES	INTEGER	The estimated raw data size of the database
LICENSE_SIZE_BYTES	INTEGER	The licensed data allowance
USAGE_PERCENT	FLOAT	Percentage of the licensed allowance used
AUDIT_START_TIMESTAMP	TIMESTAMPTZ	When the audit started
AUDIT_END_TIMESTAMP	TIMESTAMPTZ	When the audit finished
CONFIDENCE_LEVEL_PERCENT	FLOAT	The confidence level of the size estimate
ERROR_TOLERANCE_PERCENT	FLOAT	The error tolerance used for the size estimate
USED_SAMPLING	BOOLEAN	Whether data was randomly sampled (if false, all of the data was analyzed)
CONFIDENCE_INTERVAL_LOWER_BOUND_BYTES	INTEGER	The lower bound of the data size estimate within the confidence level
CONFIDENCE_INTERVAL_UPPER_BOUND_BYTES	INTEGER	The upper bound of the data size estimate within the confidence level
SAMPLE_COUNT	INTEGER	The number of data samples used to generate the estimate

Column Name	Data Type	Description
CELL_COUNT	INTEGER	The number of cells in the database
AUDITED_DATA	VARCHAR	The type of data audited, which includes regular (non-flex), flex, and total data

LICENSES

For all licenses, provides information on license types, the dates for which licenses are valid, and the limits the licenses impose.

Column Name	Data Type	Description
LICENSE_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the license.
NAME	VARCHAR	The license's name. (The license name in this column could be represented by a long license key.)
LICENSEE	VARCHAR	The entity to which the product is licensed.
START_DATE	VARCHAR	The start date for which the license is valid.
END_DATE	VARCHAR	The end date until which the license is valid (or "Perpetual" if the license has no expiration).
LICENSETYPE	VARCHAR	The type of the license (for example, Premium Edition).
PARENT	VARCHAR	The parent license (field is blank if there is no parent).
SIZE	VARCHAR	The size limit for data on the license.
IS_SIZE_LIMIT_ENFORCED	BOOLEAN	Indicates whether the license includes enforcement of data and node limits, where t is true and f is false.
NODE_RESTRICTION	VARCHAR	The node limit the license imposes.
CONFIGURED_ID	INTEGER	A long license key.

MATERIALIZED_FLEXTABLE_COLUMNS_RESULTS

Returns the results after you run the flex table function, [MATERIALIZED_FLEXTABLE_COLUMNS](#). The system table includes the following information:

Column Name	Data Type	Description
TABLE_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the license.
TABLE_SCHEMA	VARCHAR	The license's name.
TABLE_NAME	VARCHAR	The Verticaproduct toward which the license is applied.
CREATION_TIME	VARCHAR	The start date for which the license is valid.
KEY_NAME	VARCHAR	The end date until which the license is valid (or "Perpetual" if the license has no expiration).
STATUS	VARCHAR	The function status, which can be one of these values: <ul style="list-style-type: none">• ADDED• EXISTS• ERROR
MESSAGE	BOOLEAN	The message associated with the status in the previous column, one of the following: <ul style="list-style-type: none">• Added successfully• Column of same name already exists in table definition• Add operation failed• No data type guess provided to add column

Example

```
=> SELECT table_name, creation_time, key_name, status, message
FROM v_catalog.materialize_flexitable_columns_results WHERE table_name = 'twitter_r';
table_name |          creation_time          |      key_name      | status | message
-----+-----+-----+-----+-----
twitter_r  | 2013-11-20 17:00:27.945484-05 | contributors      | ADDED  | Added successfully
twitter_r  | 2013-11-20 17:00:27.94551-05  | entities.hashtags | ADDED  | Added successfully
twitter_r  | 2013-11-20 17:00:27.945519-05 | entities.urls     | ADDED  | Added successfully
twitter_r  | 2013-11-20 17:00:27.945532-05 | created_at       | EXISTS | Column of same name already
exists in table definition
(4 rows)
```

MODELS

Lists details about the models in the database.

Column Name	Data Type	Description
MODEL_ID	INTEGER	The model's internal ID.
MODEL_NAME	VARCHAR(128)	The name of the model.
SCHEMA_ID	INTEGER	The schema's internal ID.
SCHEMA_NAME	VARCHAR(128)	The name of the schema.
OWNER_ID	INTEGER	The model owner's ID.
OWNER_NAME	VARCHAR(128)	The user who created the model.
CATEGORY	VARCHAR(128)	The type of model. By default, models created in Vertica are assigned to the Vertica_Models category.
MODEL_TYPE	VARCHAR(128)	The type of algorithm used to create the model.
IS_COMPLETE	VARCHAR(128)	
CREATE_TIME	TIMESTAMPTZ	The time the model was created.
SIZE	INT	The size of the model in bytes.

NODES

Lists details about the nodes in the database.

Column Name	Data Type	Description
NODE_NAME	VARCHAR(128)	The name of the node.
NODE_ID	INT	A unique numeric ID assigned by the Vertica catalog, which identifies the node.
NODE_STATE	VARCHAR(128)	The node's current state (up, down, recovering, etc.).
NODE_ADDRESS	VARCHAR(80)	The host address of the node.
NODE_ADDRESS_FAMILY	VARCHAR(10)	The IP Version of the node_address. For example, ipv4.
EXPORT_ADDRESS	VARCHAR(8192)	The IP address of the node (on the public network) used for import/export operations and native load-balancing.
EXPORT_ADDRESS_FAMILY	VARCHAR(10)	The IP Version of the export_address. For example, ipv4.
CATALOG_PATH	VARCHAR(8192)	The absolute path to the catalog on the node.
NODE_TYPE	VARCHAR(9)	The type of the node. For more information on the types of nodes, refer to Setting Node Type .
IS_EPHEMERAL	BOOLEAN	(Deprecated) True if this node has been marked as ephemeral. (in preparation for removing it from the cluster).
STANDING_IN_FOR	VARCHAR(128)	The name of the node that this node is currently replacing.
LAST_MSG_FROM_NODE_AT	TIMESTAMPZ	The date and time the last message was received from this node.
NODE_DOWN_SINCE	TIMESTAMPZ	The amount of time that the replaced node has been unavailable.

ODBC_COLUMNS

Provides table column information. The format is defined by the ODBC standard for the ODBC SQLColumns metadata. Details on the ODBC SQLColumns format are available in the ODBC specification: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms711683%28v=vs.85%29.aspx>.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema in which the column resides. If the column does not reside in a schema, this field is empty.
TABLE_NAME	VARCHAR	The name of the table in which the column resides.
COLUMN_NAME	VARCHAR	The name of the column.
DATA_TYPE	INTEGER	The data type of the column. This can be an ODBC SQL data type or a driver-specific SQL data type. This column corresponds to the ODBC_TYPE column in the TYPES table.
DATA_TYPE_NAME	VARCHAR	The driver-specific data type name.
COLUMN_SIZE	INTEGER	The ODBC-defined data size of the column.
BUFFER_LENGTH	INTEGER	The transfer octet length of a column is the maximum number of bytes returned to the application when data is transferred to its default C data type. See http://msdn.microsoft.com/en-us/library/windows/desktop/ms713979%28v=vs.85%29.aspx
DECIMAL_DIGITS	INTEGER	The total number of significant digits to the right of the decimal point. This value has no meaning for non-decimal data types.
NUM_PREC_RADIX	INTEGER	The radix Vertica reports decimal_digits and columns_size as. This value is always 10, because it refers to a number of decimal digits, rather than a number of bits.
NULLABLE	BOOLEAN	Indicates whether the column can contain null values. Values are 0 or 1.

REMARKS	VARCHAR	The textual remarks for the column.
COLUMN_DEFAULT	VARCHAR	The default value of the column.
SQL_TYPE_ID	INTEGER	The SQL data type of the column.
SQL_DATETIME_SUB	VARCHAR	The subtype for a datetime data type. This value has no meaning for non-datetime data types.
CHAR_OCTET_LENGTH	INTEGER	The maximum length of a string or binary data column.
ORDINAL_POSITION	INTEGER	Indicates the position of the column in the table definition.
IS_NULLABLE	VARCHAR	Values can be YES or NO, determined by the value of the NULLABLE column.
IS_IDENTITY	BOOLEAN	Indicates whether the column is a sequence, for example, an auto-increment column.

PASSWORD_AUDITOR

Stores information about individual users and their password information. This table also indicates if users are using hash authentication, which is the associated security algorithm.

Column Name	Data Type	Description
USER_ID	INTEGER	Unique ID for the user.
USER_NAME	VARCHAR	Name of the user.
ACCTEXPIRED	BOOLEAN	Indicates if the user's password expires. 'f' indicates that it does not expire. 't' indicates that it does expire.
SECURITY_ALGORITHM	VARCHAR	User-level security algorithm for hash authentication. Valid values: <ul style="list-style-type: none"> 'NONE' (Default. System-level security

Column Name	Data Type	Description
		algorithm is used.) <ul style="list-style-type: none"> 'MD5' 'SHA512'
SYSTEM_SECURITY_ALGORITHM	VARCHAR	System-level security algorithm for hash authentication. Default value: 'NONE' (Uses MD5 algorithm.) Valid values: <ul style="list-style-type: none"> 'MD5' 'SHA512'
EFFECTIVE_SECURITY_ALGORITHM	VARCHAR	The resulting security algorithm, depending on the values of SECURITY_ALGORITHM and SYSTEM_SECURITY_ALGORITHM.

PASSWORDS

Contains user passwords information. This table stores current passwords and past passwords if any [Profiles](#) have PASSWORD_REUSE_TIME or PASSWORD_REUSE_MAX parameters set. See [CREATE PROFILE](#) for details.

Column Name	Data Type	Description
USER_ID	INTEGER	The ID of the user who owns the password.
USER_NAME	VARCHAR	The name of the user who owns the password.
PASSWORD	VARCHAR	The encrypted password.
PASSWORD_CREATE_TIME	DATETIME	The date and time when the password was created.
IS_CURRENT_PASSWORD	BOOLEAN	Denotes whether this is the user's current password. Non-current passwords are retained to enforce password reuse limitations.

Column Name	Data Type	Description
PROFILE_ID	INTEGER	The ID number of the profile to which the user is assigned.
PROFILE_NAME	VARCHAR	The name of the profile to which the user is assigned.
PASSWORD_REUSE_MAX	VARCHAR	The number password changes that must take place before an old password can be reused.
PASSWORD_REUSE_TIME	VARCHAR	The amount of time that must pass before an old password can be reused.

PRIMARY_KEYS

Provides primary key information.

Column Name	Data Type	Description
CONSTRAINT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
CONSTRAINT_NAME	VARCHAR	The constraint name for which information is listed.
COLUMN_NAME	VARCHAR	The column name for which information is listed.
ORDINAL_POSITION	VARCHAR	The position of the column within the key. The numbering of columns starts at 1.
TABLE_NAME	VARCHAR	The table name for which information is listed.
CONSTRAINT_TYPE	VARCHAR	The constraint type, p, for primary key.
IS_ENABLED	BOOLEAN	Indicates if a table column constraint for a PRIMARY KEY is enabled by default. Can be t (True) or f (False).
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.

PROFILE_PARAMETERS

Defines what information is stored in profiles.

Column Name	Data Type	Description
PROFILE_ID	INTEGER	The ID of the profile to which this parameter belongs.
PROFILE_NAME	VARCHAR	The name of the profile to which this parameter belongs.
PARAMETER_TYPE	VARCHAR	The policy type of this parameter (<code>password_complexity</code> , <code>password_security</code> , etc.)
PARAMETER_NAME	VARCHAR	The name of the parameter.
PARAMETER_LIMIT	VARCHAR	The parameter's value.

PROFILES

Provides information about password policies that you set using the [CREATE PROFILE](#) statement.

Column Name	Data Type	Description
PROFILE_ID	INTEGER	Unique identifier for the profile.
PROFILE_NAME	VARCHAR	Profile name.
PASSWORD_LIFE_TIME	VARCHAR	Number of days before the user's password expires. After expiration, the user is forced to change passwords during login or warned that their password has expired if <code>password_grace_time</code> is set to a value other than zero or unlimited.
PASSWORD_GRACE_TIME	VARCHAR	Number of days users are allowed to log in after their passwords expire. During the grace time, users are warned about their expired passwords when they log in. After the grace period, the user is forced to change passwords if he or she hasn't already.
PASSWORD_REUSE_MAX	VARCHAR	Number of password changes that must occur before the current password can be reused.
PASSWORD_REUSE_TIME	VARCHAR	Number of days that must pass after setting a password before it can be used again.

Column Name	Data Type	Description
FAILED_LOGIN_ATTEMPTS	VARCHAR	Number of consecutive failed login attempts that triggers Vertica to lock the account.
PASSWORD_LOCK_TIME	VARCHAR	Number of days an account is locked after being locked due to too many failed login attempts.
PASSWORD_MAX_LENGTH	VARCHAR	Maximum number of characters allowed in a password.
PASSWORD_MIN_LENGTH	VARCHAR	Minimum number of characters required in a password.
PASSWORD_MIN_LETTERS	VARCHAR	The minimum number of letters (either uppercase or lowercase) required in a password.
PASSWORD_MIN_LOWERCASE_LETTERS	VARCHAR	The minimum number of lowercase.
PASSWORD_MIN_UPPERCASE_LETTERS	VARCHAR	The minimum number of uppercase letters required in a password.
PASSWORD_MIN_DIGITS	VARCHAR	The minimum number of digits required in a password.
PASSWORD_MIN_SYMBOLS	VARCHAR	The minimum of symbols (for example, !, #, \$, etc.) required in a password.

Notes

Non-superusers querying this table see only the information for the profile to which they are assigned.

See Also

- [CREATE PROFILE](#)
- [ALTER PROFILE](#)

PROJECTION_CHECKPOINT_EPOCHS

Records when each projection checkpoint epoch changes.

Column Name	Data Type	Description
NODE_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the node for which information is listed
NODE_NAME	VARCHAR	Node name for which information is listed.
PROJECTION_SCHEMA_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the specific schema that contains the projection.
PROJECTION_SCHEMA	VARCHAR	Schema containing the projection.
PROJECTION_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	Projection name for which information is listed.
IS_UP_TO_DATE	BOOLEAN	Indicates whether the projection is up to date, where <i>t</i> is true and <i>f</i> is false. Projections must be up to date for queries to use them.
CHECKPOINT_EPOCH	INTEGER	Checkpoint epoch of the projection on the corresponding node. Data up to and including this epoch is in persistent storage. If the node were to fail, without moving more data out of the WOS, data after this epoch would need to be recovered.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Examples

```
=> SELECT epoch FROM t;
epoch
-----
    52
    52
    53
(3 rows)

=> SELECT node_name, projection_schema, projection_name, is_up_to_date, checkpoint_epoch
FROM projection_checkpoint_epochs;
node_name          | projection_schema | projection_name | is_up_to_date | checkpoint_epoch
-----+-----+-----+-----+-----
v_vmart_node0001  | public           | t_super        | t              | 51
v_vmart_node0001  | public           | p_super        | t              | 51
(2 rows)

=> SELECT DO_TM_TASK('moveout','');
              do_tm_task
-----
Task: moveout
(Table: public.t) (Projection: public.t_super)
(Table: public.p) (Projection: public.p_super)
(1 row)

=> SELECT node_name, projection_schema, projection_name, is_up_to_date, checkpoint_epoch
FROM projection_checkpoint_epochs;
node_name          | projection_schema | projection_name | is_up_to_date | checkpoint_epoch
-----+-----+-----+-----+-----
v_vmart_node0001  | public           | t_super        | t              | 53
v_vmart_node0001  | public           | p_super        | t              | 53
(2 rows)
```

PROJECTION_COLUMNS

Provides information about projection columns, such as encoding type, sort order, type of statistics, and the time at which columns statistics were last updated.

Column Name	Data Type	Description
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
PROJECTION_COLUMN_	VARCHAR	The projection column name.

Column Name	Data Type	Description
NAME		
COLUMN_POSITION	INTEGER	The ordinal position of a projection's column used in the CREATE PROJECTION statement.
SORT_POSITION	INTEGER	The projection's column sort specification, as specified in <code>CREATE PROJECTION . . ORDER BY</code> clause. If the column is not included in the projection's sort order, <code>SORT_POSITION</code> output is NULL.
COLUMN_ID	INTEGER	A unique numeric object ID (OID) that identifies the associated projection column object and is assigned by the Vertica catalog. This field is helpful as a key to other system tables.
DATA_TYPE	VARCHAR	Matches the corresponding table column data type (see V_CATALOG.COLUMNS). <code>DATA_TYPE</code> is provided as a complement to <code>ENCODING_TYPE</code> .
ENCODING_TYPE	VARCHAR	The encoding type defined on the projection column.
ACCESS_RANK	INTEGER	The access rank of the projection column. See the <code>ACCESSRANK</code> parameter in the CREATE PROJECTION statement for more information.
GROUP_ID	INTEGER	A unique numeric ID (OID) that identifies the group and is assigned by the Vertica catalog.
TABLE_SCHEMA	VARCHAR	The name of the schema in which the projection is stored.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the table.
TABLE_NAME	VARCHAR	The table name that contains the

Column Name	Data Type	Description
		projection.
TABLE_COLUMN_ID	VARCHAR	A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.
TABLE_COLUMN_NAME	VARCHAR	The projection's corresponding table column name.
STATISTICS_TYPE	VARCHAR	The type of statistics the column contains: <ul style="list-style-type: none"> • NONE: No statistics • ROWCOUNT: Created from existing catalog metadata, which Vertica automatically and periodically updates • FULL: Created by running ANALYZE_STATISTICS
STATISTICS_UPDATED_TIMESTAMP	TIMESTAMPTZ	The time at which the columns statistics were last updated. By querying this column, along with STATISTICS_TYPE and PROJECTION_COLUMN_NAME, you can identify projection columns whose statistics need updating. See also V_CATALOG.PROJECTIONS.HAS_STATISTICS .
IS_EXPRESSION	BOOLEAN	Indicates whether this projection column is calculated with an expression. For aggregate columns, IS_EXPRESSION is always true.
IS_AGGREGATE	BOOLEAN	Indicates whether the column is an aggregated column in a live aggregate projection. IS_AGGREGATE is always false for Top-K projection columns.
PARTITION_BY_POSITION	INTEGER	Position of that column in the PARTITION BY and GROUP BY clauses, if applicable.

Column Name	Data Type	Description
ORDER_BY_POSITION	INTEGER	Set only for Top-K projections , specifies the column's position in the ORDER BY clause, as defined in the projection definition's window partition clause . If the column is omitted from the ORDER BY clause, ORDER_BY_POSITION output is NULL.
ORDER_BY_TYPE	INTEGER	Type of sort order: <ul style="list-style-type: none">• ASC NULLS FIRST• ASC NULLS LAST• DESC NULLS FIRST• DESC NULLS LAST
COLUMN_EXPRESSION	VARCHAR	Expression that calculates the column value.

Examples

See [Determining When Statistics Were Last Updated](#).

See Also

- [PROJECTIONS](#)
- [ANALYZE_STATISTICS](#)
- [CREATE PROJECTION](#)
- [Collecting Database Statistics](#)

PROJECTION_DELETE_CONCERNS

Lists projections whose design may cause performance issues when deleting data. This table is generated by calling the [EVALUATE_DELETE_PERFORMANCE](#) function. See [DELETE and UPDATE Optimization](#) in the Administrator's Guide for more information.

Column Name	Data Type	Description
PROJECTION_ID	INTEGER	The ID number of the projection
PROJECTION_SCHEMA	VARCHAR	The schema containing the projection
PROJECTION_NAME	VARCHAR	The projection's name
CREATION_TIME	TIMESTAMPTZ	When the projection was created
LAST_MODIFIED_TIME	TIMESTAMPTZ	When the projection was last modified
COMMENT	VARCHAR	A comment describing the potential delete performance issue.

PROJECTIONS

Provides information about projections.

Column Name	Data Type	Description
PROJECTION_SCHEMA_ID	INTEGER	A unique numeric ID that identifies the specific schema that contains the projection and is assigned by the Vertica catalog.
PROJECTION_SCHEMA	VARCHAR	The name of the schema that contains the projection.
PROJECTION_ID	INTEGER	A unique numeric ID that identifies the projection and is assigned by the Vertica catalog.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
PROJECTION_BASENAME	VARCHAR	The base name used for other projections. For auto-created projections, <code>projection_basename</code> is identical to the <code>anchor_table_name</code> . For a manually-created projection, <code>projection_basename</code> is the name you supply.
OWNER_ID	INTEGER	A unique numeric ID that identifies the projection owner and is

ID	ER	assigned by the Vertica catalog.
OWNER_NAME	VARCHAR	The name of the projection's owner.
ANCHOR_TABLE_ID	INTEGER	For pre-join projections, the unique numeric identification (OID) of the anchor table. If the projection is not a pre-join projection, this value is the OID of the table from which the projection was created. A projection has only one anchor (fact) table.
ANCHOR_TABLE_NAME	VARCHAR	For pre-join projections, the name of the anchor table. If the projection is not a pre-join projection, the name of the table from which the projection was created.
NODE_ID	INTEGER	A unique numeric ID (OID) for any nodes that contain any unsegmented projections.
NODE_NAME	VARCHAR	The names of any nodes that contain the projection. This column returns information for unsegmented projections only.
IS_PREJOIN	BOOLEAN	Specifies whether the projection is a pre-join projection, where <i>t</i> is true and <i>f</i> is false.
CREATE_D_EPOCH	INTEGER	The epoch in which the projection was created.
CREATE_TYPE	VARCHAR	The method in which the projection was created: <ul style="list-style-type: none"> • <code>CREATE PROJECTION</code>: A custom projection created using a <code>CREATE PROJECTION</code> statement. • <code>CREATE TABLE</code>: A superprojection that was automatically created when its associated table was created using a <code>CREATE TABLE</code> statement. • <code>ALTER TABLE</code>: The system automatically created the key projection in response to a non-empty table. • <code>CREATE TABLE WITH PROJ CLAUSE</code>: A superprojection created using a <code>CREATE TABLE</code> statement. • <code>DELAYED_CREATION</code>: A superprojection that was automatically created when data was loaded into its associated table.

		<ul style="list-style-type: none"> • DESIGNER: A new projection created by the Database Designer. • SYSTEM TABLE: A projection that was automatically created for a system table. <p>Rebalancing does not change the <code>CREATE_TYPE</code> value for a projection.</p>
<code>VERIFY_FAULT_TOLERANCE</code>	<code>INTEGER</code>	The projection K-safe value. This value can be greater than the database K-safety value (if more replications of a projection exist than are required to meet the database K-safety). This value cannot be less than the database K-safe setting.
<code>IS_UP_TO_DATE</code>	<code>BOOLEAN</code>	Specifies whether the projection is up to date. Projections must be up to date to be used in queries.
<code>HAS_STATISTICS</code>	<code>BOOLEAN</code>	Specifies whether there are statistics for any column in the projection: <ul style="list-style-type: none"> • This column returns true only when all non-epoch columns for a table have full statistics. Otherwise, the column returns false. See ANALYZE_STATISTICS(). • Projections that have no data never have full statistics. Use the PROJECTION_STORAGE system table to determine whether your projection contains data.
<code>IS_SEGMENTED</code>	<code>BOOLEAN</code>	Specifies whether the projection is segmented.
<code>SEGMENT_EXPRESSION</code>	<code>VARCHAR</code>	The segmentation expression used for the projection. In the following example for the <code>clicks_agg</code> projection, the values <code>hash(clicks.user_id, (clicks.click_time)::date)</code> indicate that the projection was created with the following expression: <pre>...segmented by hash(clicks.user_id, (clicks.click_time)::date)</pre>
<code>SEGMENT_RANGE</code>	<code>VARCHAR</code>	The percentage of projection data stored on each node, according to the segmentation expression. For example, segmenting a projection by <code>HASH()</code> on all nodes results in a <code>segment_range</code> value such as the following: <pre>segment_range implicit range: v_testcr_node0005[33.3%] v_testcr_</pre>

		node0006[33.3%] v_testcr_node0004[33.3%]
IS_SUPER_PROJECTION	BOOLEAN	Specifies whether a projection is a superprojection.
AGGREGATE_TYPE	VARCHAR	Specifies the type of live aggregate projection: <ul style="list-style-type: none"> GROUPBY TOPK
IS_KEY_CONSTRAINT_PROJECTION	BOOLEAN	Indicates whether a projection is a key constraint projection: <ul style="list-style-type: none"> t: A key constraint projection that validates a key constraint. (Vertica uses the projection to efficiently enforce at least one enabled key constraint.) f: Not a projection that validates a key constraint.
HAS_EXPRESSIONS	BOOLEAN	Specifies whether this projection has expressions that define the column values. HAS_EXPRESSIONS is always true for live aggregate projections.
IS_AGGREGATE_PROJECTION	BOOLEAN	Specifies whether this projection is a live aggregate projection.

See Also

- [ANALYZE_STATISTICS](#)
- [PROJECTION_COLUMNS](#)
- [PROJECTION_STORAGE](#)

RESOURCE_POOL_DEFAULTS

Provides information about the default parameter values for built-in and user-defined resource pools. You can use [ALTER RESOURCE POOL](#) to restore default parameter values for any

resource pool by setting that parameter to DEFAULT. To see the default values for the built-in resource pools, refer to [Built-In Pool Configuration](#).

Privileges

None

See Also

- [RESOURCE_POOLS](#)

RESOURCE_POOLS

Displays information about the parameters specified for the resource pool by [CREATE RESOURCE POOL](#) or [ALTER RESOURCE POOL](#).

Note: Column names in the RESOURCE_POOL table mirror syntax in the [CREATE RESOURCE POOL](#) statement. Therefore, column names do not use underscores.

Column Name	Data Type	Description
NAME	VARCHAR	The name of the resource pool.
IS_INTERNAL	BOOLEAN	Denotes whether a pool is one of the Built-In Pools .
MEMORYSIZE	VARCHAR	Value of the amount of memory allocated to the resource pool.
MAXMEMORYSIZE	VARCHAR	Value assigned as the maximum size the resource pool could grow by borrowing memory from the GENERAL pool.
EXECUTIONPARALLELISM	INTEGER	Limits the number of threads used to process any single query issued in this resource pool
PRIORITY	INTEGER	Value of PRIORITY parameter specified when defining the pool.

Column Name	Data Type	Description
RUNTIMEPRIORITY	VARCHAR	<p>Value that indicates the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are:</p> <ul style="list-style-type: none"> • HIGH • MEDIUM (default) • LOW <p>These values are relative to each other. Queries with a HIGH run-time priority are given more CPU and I/O resources than those with a MEDIUM or LOW run-time priority.</p>
RUNTIMEPRIORITYTHRESHOLD	INTEGER	Limits in seconds how soon a query must finish before the Resource Manager assigns to it the resource pool's RUNTIMEPRIORITY setting.
QUEUETIMEOUT	INTEGER or INTERVAL	The interval or value in seconds of the QUEUETIMEOUT parameter specified when defining the pool. Represents the maximum amount of time the request is allowed to wait for resources to become available before being rejected.
PLANNEDCONCURRENCY	INTEGER	Value of PLANNEDCONCURRENCY parameter specified when defining the pool, which represents the preferred number of concurrently executing queries in the resource pool.
MAXCONCURRENCY	INTEGER	Value of MAXCONCURRENCY parameter specified when defining the pool, which represents the maximum number of concurrent execution slots available to the resource pool.

Column Name	Data Type	Description
RUNTIMECAP	INTERVAL	The maximum time a query in the pool can execute.
SINGLEINITIATOR	BOOLEAN	Specifies whether all requests using this pool are issued against the same initiator node or multiple initiator nodes can be used. Included for backwards compatibility. For all built-in resource pools, this must be set to <code>false</code> for all user-defined pools.
CPUAFFINITYSET	VARCHAR	Value which represents the set of CPUs on which queries associated with this pool are executed. For example, '0, 2-4' for the 0, 2, 3, and 4 CPUs, or '25%' for a percentage of available CPUs. Percentage values are rounded down to whole CPUs.
CPUAFFINITYMODE	VARCHAR	The mode of the CPU affinity. If CPUAFFINITYSET is set to a CPU percentage or index/index list, then its value is one of the following: <ul style="list-style-type: none"> • SHARED: Pool is pinned to the CPU indexes or percentage defined in CPUAFFINITYSET, and other pools can share the same CPUs • EXCLUSIVE: Pool is pinned to the CPU indexes or percentage defined in CPUAFFINITYSET, but other pools cannot use the same CPUs. • ANY: Pool has no affinity for a specific CPU or percentage of available CPUs.
CASCADETO	VARCHAR	The name of the secondary resource pool, if one exists.

See Also

- [CREATE RESOURCE POOL](#)
- [ALTER RESOURCE POOL](#)

ROLES

Contains the names of all roles the user can access, along with any roles that have been assigned to those roles.

Tip: You can also use the function [HAS_ROLE](#) to see if a role is available to a user.

Column Name	Data Type	Description
ASSIGNED_ROLES	VARCHAR	The names of any roles that have been granted to this role. By enabling the role, the user also has access to the privileges of these additional roles. Note: If you see an asterisk in the ASSIGNED_ROLES column output, it means the user has roles WITH ADMIN OPTION.
NAME	VARCHAR	The name of a role that the user can access.
ROLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the role.
LDAP_DN	VARCHAR	Indicates whether or not the Vertica Analytic Database role maps to an LDAP Link group. When the column is set to dn, the Vertica role maps to LDAP Link.
LDAP_URI_HASH	VARCHAR	The URI hash number for the LDAP role.
IS_ORPHANED_FROM_LDAP	VARCHAR	Indicates if the role is disconnected (orphaned) from LDAP, valid values are: t - role is orphaned f - role is not orphaned For more information see Troubleshooting LDAP Link

Column Name	Data Type	Description
		Issues

See Also

- [GRANTS](#)
- [HAS_ROLE](#)
- [USERS](#)

SCHEMATA

Provides information about schemas in the database.

Column Name	Data Type	Description
SCHEMA_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the specific schema.
SCHEMA_NAME	VARCHAR	Schema name for which information is listed.
SCHEMA_OWNER_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the owner who created the schema.
SCHEMA_OWNER	VARCHAR	Name of the owner who created the schema.
SYSTEM_SCHEMA_CREATOR	VARCHAR	Creator information for system schema or NULL for non-system schema
CREATE_TIME	TIMESTAMPTZ	Time when the schema was created.
IS_SYSTEM_SCHEMA	BOOLEAN	Indicates whether the schema was created for system use, where <i>t</i> is true and <i>f</i> is false.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

SEQUENCES

Displays information about the parameters specified for a sequence using the [CREATE SEQUENCE](#) statement.

Column Name	Data Type	Description
SEQUENCE_SCHEMA	VARCHAR	Schema in which the sequence was created.
SEQUENCE_NAME	VARCHAR	Name of the sequence defined in the CREATE SEQUENCE statement.
OWNER_NAME	VARCHAR	Name of the owner; for example, dbadmin.
IDENTITY_TABLE_NAME	VARCHAR	If created by an <code>auto_increment</code> or <code>identity</code> column, the name of the table to which it belongs. See column constraints in the CREATE TABLE statement.
SESSION_CACHE_COUNT	INTEGER	Count of values cached in a session.
ALLOW_CYCLE	BOOLEAN	Values allowed to cycle when max/min is reached. See <code>CYCLE NO CYCLE</code> parameter in CREATE SEQUENCE .
OUTPUT_ORDERED	BOOLEAN	Values guaranteed to be ordered (always false).
INCREMENT_BY	INTEGER	Sequence values are incremented by this number (negative for reverse sequences).
MINIMUM	INTEGER	Minimum value the sequence can generate.
MAXIMUM	INTEGER	Maximum value the sequence can generate.
CURRENT_VALUE	INTEGER	Specifies how many sequence numbers Vertica has distributed to the nodes in your cluster. Includes all nodes.
SEQUENCE_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the schema.
SEQUENCE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the sequence.
OWNER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user who created the sequence.

Column Name	Data Type	Description
IDENTITY_TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table to which the column belongs (if created by an <code>auto_increment</code> or <code>identity</code> column).

Examples

Create a simple sequence:

```
=> CREATE SEQUENCE my_seq MAXVALUE 5000 START 150;
CREATE SEQUENCE
```

Return information about the sequence you just created:

```
=> \x
Expanded display is on.
=> SELECT * FROM sequences;
-[ RECORD 1 ]-----+-----
sequence_schema | public
sequence_name   | my_seq
owner_name      | dbadmin
identity_table_name |
session_cache_count | 250000
allow_cycle     | f
output_ordered  | f
increment_by    | 1
minimum         | 1
maximum         | 5000
current_value   | 149
sequence_schema_id | 45035996273704966
sequence_id     | 45035996273844996
owner_id        | 45035996273704962
identity_table_id | 0
```

An identity column is a sequence available only for numeric column types. To identify what column in a table, if any, is an identity column, search the `COLUMNS` table to find the identity column in a table:

```
=> CREATE TABLE testid (c1 IDENTITY(1, 1, 1000), c2 INT);
=> \x
Expanded display is on.
=> SELECT * FROM COLUMNS WHERE is_identity='t' AND table_name='testid';
-[ RECORD 1 ]-----+-----
table_id          | 45035996274150730
table_schema     | public
table_name       | testid
is_system_table  | f
column_name      | c1
data_type        | int
```

```
data_type_id          | 6
data_type_length      | 8
character_maximum_length |
numeric_precision     |
numeric_scale         |
datetime_precision    |
interval_precision    |
ordinal_position      | 1
is_nullable           | f
column_default        |
is_identity           | t
```

Use the SEQUENCES table to get detailed information about the sequence in testid:

```
=> SELECT * FROM sequences WHERE identity_table_name='testid';
-[ RECORD 1 ]-----+-----
sequence_schema      | public
sequence_name        | testid_c1_seq
owner_name           | dbadmin
identity_table_name  | testid
session_cache_count  | 1000
allow_cycle          | f
output_ordered       | f
increment_by         | 1
minimum              | 1
maximum              | 9223372036854775807
current_value        | 0
sequence_schema_id   | 45035996273704976
sequence_id          | 45035996274150770
owner_id             | 45035996273704962
identity_table_id    | 45035996274150768
```

Use the vsql command \ds to return a list of sequences. The following results show the two sequences created in the preceding examples. If more sequences existed, the table would list them.

The CurrentValue of the new sequence is one less than the start number you specified in the CREATE SEQUENCE and IDENTITY commands, because you have not yet used NEXTVAL to instantiate the sequences to assign their cache or supply their first start values.

```
=> \ds
List of Sequences
-[ RECORD 1 ]-----+-----
Schema      | public
Sequence    | my_seq
CurrentValue | 149
IncrementBy | 1
Minimum     | 1
Maximum     | 5000
AllowCycle  | f
Comment     |
-[ RECORD 2 ]-----+-----
Schema      | public
Sequence    | testid_c1_seq
CurrentValue | 0
```

```

IncrementBy | 1
Minimum     | 1
Maximum     | 9223372036854775807
AllowCycle  | f
Comment     |

```

See Also

- [CREATE SEQUENCE](#)

STORAGE_LOCATIONS

Provides information about storage locations, their IDs labels, and status.

Column Name	Data Type	Description
LOCATION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the storage location.
NODE_NAME	VARCHAR	The node name on which the storage location exists.
LOCATION_PATH	VARCHAR	The path where the storage location is mounted.
LOCATION_USAGE	VARCHAR	The type of information stored in the location: <ul style="list-style-type: none"> • DATA: Only data is stored in the location. • TEMP: Only temporary files that are created during loads or queries are stored in the location. • DATA,TEMP: Both types of files are stored in the location. • USER: The storage location can be used by non-dbadmin users, who are granted access to the storage location • CATALOG: The area is used for the Vertica catalog. This usage is set internally and cannot be removed or changed.
IS_RETIRED	BOOLEAN	Whether the storage location has been retired. This column has a value of <code>t</code> (<code>true</code>) if the location is retired, or <code>f</code> (<code>false</code>) if it is not.
LOCATION_LABEL	VARCHAR	The label associated with a specific storage location, added

Column Name	Data Type	Description
		with the ALTER_LOCATION_LABEL function.
RANK	INTEGER	The Access Rank value either assigned or supplied to the storage location, as described in Prioritizing Column Access Speed .
THROUGHPUT	INTEGER	The throughput performance of the storage location, measured in MB/sec. You can get location performance values using MEASURE_LOCATION_PERFORMANCE , and set them with the SET_LOCATION_PERFORMANCE function.
LATENCY	INTEGER	The measured latency of the storage location as number of data seeks per second. You can get location performance values using MEASURE_LOCATION_PERFORMANCE , and set them with the SET_LOCATION_PERFORMANCE function.

Privileges

Must be a superuser.

See Also

- [DISK_STORAGE](#)
- [MEASURE_LOCATION_PERFORMANCE](#)
- [SET_LOCATION_PERFORMANCE](#)
- [STORAGE_POLICIES](#)
- [STORAGE_USAGE](#)
- [Storage Management Functions](#)

SYSTEM_COLUMNS

Provides table column information for [SYSTEM_TABLES](#).

Column Name	Data Type	Description
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
TABLE_NAME	VARCHAR	The table name for which information is listed.
IS_SYSTEM_TABLE	BOOLEAN	Indicates whether the table is a system table, where <i>t</i> is true and <i>f</i> is false.
COLUMN_ID	VARCHAR	A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.
COLUMN_NAME	VARCHAR	The column name for which information is listed in the database.
DATA_TYPE	VARCHAR	The data type assigned to the column; for example VARCHAR(16).
DATA_TYPE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the data type.
DATA_TYPE_LENGTH	INTEGER	The maximum allowable length of the data type.
CHARACTER_MAXIMUM_LENGTH	INTEGER	The maximum allowable length of the column.
NUMERIC_PRECISION	INTEGER	The number of significant decimal digits.
NUMERIC_SCALE	INTEGER	The number of fractional digits.
DATETIME_PRECISION	INTEGER	For TIMESTAMP data type, returns the declared precision; returns null if no precision was declared.
INTERVAL_PRECISION	INTEGER	The number of fractional digits retained in the seconds field.
ORDINAL_POSITION	INTEGER	The position of the column relative to other columns in the table.
IS_NULLABLE	BOOLEAN	Indicates whether the column can contain null values, where <i>t</i> is true and <i>f</i> is false.
COLUMN_DEFAULT	VARCHAR	The default value of a column, such as empty or expression.

SYSTEM_TABLES

Returns a list of all system table names.

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the schema.
TABLE_SCHEMA	VARCHAR	The schema name in which the system table resides. Values are: <ul style="list-style-type: none"> V_CATALOG V_MONITOR.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table.
TABLE_NAME	VARCHAR	The name of the system table.
TABLE_DESCRIPTION	VARCHAR	A description of the system table's purpose.
IS_SUPERUSER_ONLY	BOOLEAN	Indicates if the table is accessible only by a user with superuser privileges. Values are: t - table is accessible by superuser only f - table is accessible by other users
IS_MONITORABLE	BOOLEAN	Indicates if the table is accessible by a user with the SYSMONITOR role enabled. Values are: t - table is accessible by SYSMONITOR role f - table is not accessible by SYSMONITOR role See SYSMONITOR Role .
IS_ACCESSIBLE_DURING_LOCKDOWN	BOOLEAN	Indicates if the table is restricted using the RestrictSystemTables parameter. See System Table Restriction .

TABLE_CONSTRAINTS

Provides information about table constraints.

Column Name	Data Type	Description
CONSTRAINT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.
CONSTRAINT_NAME	VARCHAR	The name of the constraint, if specified as UNIQUE, FOREIGN KEY, NOT NULL, PRIMARY KEY, or CHECK.
CONSTRAINT_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the schema containing the constraint.
CONSTRAINT_KEY_COUNT	INTEGER	The number of constraint keys.
FOREIGN_KEY_COUNT	INTEGER	The number of foreign keys.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the table.
TABLE_NAME	VARCHAR	The name of the table that contains the UNIQUE, FOREIGN KEY, NOT NULL, or PRIMARY KEY constraint
FOREIGN_TABLE_ID	INTEGER	The unique object ID of the foreign table referenced in a foreign key constraint (zero if not a foreign key constraint).
CONSTRAINT_TYPE	CHAR	Indicates the constraint type. Valid Values: <ul style="list-style-type: none"> • c — check • f — foreign • p — primary • u — unique
IS_ENABLED	BOOLEAN	Indicates if a constraint for a primary key, unique key, or check constraint is currently enabled. Can be t (True) or f (False).

Column Name	Data Type	Description
PREDICATE	VARCHAR	For check constraints, the SQL expression.

See Also

[ANALYZE_CONSTRAINTS](#)

TABLES

Provides information about all tables in the database.

Tip: Columns TABLE_SCHEMA and TABLE_NAME are case sensitive. To query TABLES on these columns, use the case-insensitive ILIKE predicate. For example:

```
SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'Store%';
```

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	A unique numeric ID that identifies the schema and is assigned by the Vertica catalog.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
TABLE_ID	INTEGER	A unique numeric ID that identifies the table and is assigned by the Vertica catalog.
TABLE_NAME	VARCHAR	The table name for which information is listed.
OWNER_ID	INTEGER	A unique numeric ID that identifies the owner and is assigned by the Vertica catalog.
OWNER_NAME	VARCHAR	The name of the user who created the table.
IS_TEMP_TABLE	BOOLEAN	Indicates whether this table is a temporary table.
IS_SYSTEM_TABLE	BOOLEAN	Indicates whether table is a system table.
FORCE_OUTER	INTEGER	Specifies whether this table is joined to

Column Name	Data Type	Description
		another as an inner or outer input. For details, see Controlling Join Inputs in Analyzing Data.
IS_FLEXTABLE	BOOLEAN	Indicates whether the table is a Flex table.
HAS_AGGREGATE_PROJECTION	BOOLEAN	Indicates whether the table has live aggregate projections.
SYSTEM_TABLE_CREATOR	VARCHAR	The name of the process that created the table, such as Designer.
PARTITION_EXPRESSION	VARCHAR	The partition expression for the table.
CREATE_TIME	TIMESTAMP	Returns the timestamp, indicating when the table was created.
TABLE_DEFINITION	VARCHAR	The COPY statement table definition. This column is applicable only to external tables.
RECOVER_PRIORITY	INTEGER	The priority rank for the table for a Recovery By Table .
STORAGE_MODE	INTEGER	Specifies the load method a table uses, set to one of the following integer values: 0: The initial value for all tables that pre-date Vertica 8.1.x 1: Direct 5: Trickle 6: Auto (default)

Examples

Find when tables were created:

```
=> SELECT table_schema, table_name, create_time FROM tables;
table_schema | table_name | create_time
-----+-----+-----
public       | customer_dimension | 2011-08-15 11:18:25.784203-04
public       | product_dimension  | 2011-08-15 11:18:25.815653-04
```

```

public      | promotion_dimension | 2011-08-15 11:18:25.850592-04
public      | date_dimension      | 2011-08-15 11:18:25.892347-04
public      | vendor_dimension    | 2011-08-15 11:18:25.942805-04
public      | employee_dimension  | 2011-08-15 11:18:25.966985-04
public      | shipping_dimension  | 2011-08-15 11:18:25.999394-04
public      | warehouse_dimension | 2011-08-15 11:18:26.461297-04
public      | inventory_fact      | 2011-08-15 11:18:26.513525-04
store       | store_dimension     | 2011-08-15 11:18:26.657409-04
store       | store_sales_fact    | 2011-08-15 11:18:26.737535-04
store       | store_orders_fact   | 2011-08-15 11:18:26.825801-04
online_sales | online_page_dimension | 2011-08-15 11:18:27.007329-04
online_sales | call_center_dimension | 2011-08-15 11:18:27.476844-04
online_sales | online_sales_fact   | 2011-08-15 11:18:27.49749-04
(15 rows)

```

Find out whether certain tables are temporary and flex tables:

```

=> SELECT distinct table_name, table_schema, is_temp_table, is_flextable FROM v_catalog.tables
    WHERE table_name ILIKE 't%';
 table_name | table_schema | is_temp_table | is_flextable
-----+-----+-----+-----
t2_temp    | public       | t             | t
tt_keys    | public       | f             | f
t2_temp_keys | public      | f             | f
t3         | public       | t             | f
t1         | public       | f             | f
t9_keys    | public       | f             | f
t2_keys    | public       | f             | t
t6         | public       | t             | f
t5         | public       | f             | f
t2         | public       | f             | t
t8         | public       | f             | f
t7         | public       | t             | f
tt         | public       | t             | t
t2_keys_keys | public      | f             | f
t9         | public       | t             | t
(15 rows)

```

TEXT_INDICES

Provides summary information about the text indices in Vertica.

Column Name	Data Type	Description
INDEX_ID	INT	A unique numeric ID that identifies the index and is assigned by the Vertica catalog.
INDEX_NAME	VARCHAR	The name of the text index.
INDEX_SCHEMA_NAME	VARCHAR	The schema name of the text index.
SOURCE_TABLE_ID	INT	A unique numeric ID that identifies the table and is

Column Name	Data Type	Description
		assigned by the Vertica catalog.
SOURCE_TABLE_NAME	VARCHAR	The name of the source table used to build the index.
SOURCE_TABLE_SCHEMA_NAME	VARCHAR	The schema name of the source table.
TOKENIZER_ID	INT	A unique numeric ID that identifies the tokenizer and is assigned by the Vertica catalog.
TOKENIZER_NAME	VARCHAR	The name of the tokenizer used when building the index.
TOKENIZER_SCHEMA_NAME	VARCHAR	The schema name of the tokenizer.
STEMMER_ID	INT	A unique numeric ID that identifies the stemmer and is assigned by the Vertica catalog.
STEMMER_NAME	VARCHAR	The name of the stemmer used when building the index.
STEMMER_SCHEMA_NAME	VARCHAR	The schema name of the stemmer.
TEXT_COL	VARCHAR	The text column used to build the index.

TYPES

Provides information about supported data types.

Column Name	Data Type	Description
TYPE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the specific data type.
ODBC_TYPE	INTEGER	The numerical ODBC type.
ODBC_SUBTYPE	INTEGER	The numerical ODBC subtype, used to differentiate types such as time and interval that have multiple subtypes.
MIN_SCALE	INTEGER	The minimum number of digits supported to the right of the decimal point for the data type.

Column Name	Data Type	Description
MAX_SCALE	INTEGER	The maximum number of digits supported to the right of the decimal point for the data type. A value of 0 is used for types that do not use decimal points.
COLUMN_SIZE	INTEGER	The number of characters required to display the type. See: http://msdn.microsoft.com/en-us/library/windows/desktop/ms711786%28v=VS.85%29.aspx for the details on COLUMN_SIZE for each type.
INTERVAL_MASK	INTEGER	For data types that are intervals, the bitmask to determine the range of the interval from the Vertica TYPE_ID. Details are available in the Vertica SDK.
TYPE_NAME	VARCHAR	The data type name associated with a particular data type ID.
CREATION_PARAMETERS	VARCHAR	A list of keywords, separated by commas, corresponding to each parameter that the application may specify in parentheses when using the name that is returned in the TYPE_NAME field. The keywords in the list can be any of the following: length, precision, or scale. They appear in the order that the syntax requires them to be used.

USER_AUDITS

Lists the results of database and object size audits generated by users calling the [AUDIT](#) function. See [Monitoring Database Size for License Compliance](#) in the Administrator's Guide for more information.

Column Name	Data Type	Description
SIZE_BYTES	INTEGER	The estimated raw data size of the database
USER_ID	INTEGER	The ID of the user who generated the audit
USER_NAME	VARCHAR	The name of the user who generated the audit
OBJECT_ID	INTEGER	The ID of the object being audited

Column Name	Data Type	Description
OBJECT_TYPE	VARCHAR	The type of object being audited (table, schema, etc.)
OBJECT_SCHEMA	VARCHAR	The schema containing the object being audited
OBJECT_NAME	VARCHAR	The name of the object being audited
AUDITED_SCHEMA_NAME	VARCHAR	The name of the schema on which you want to query HISTORICAL data. After running audit on a table, you can drop the table. In this case, object_schema becomes NULL.
AUDITED_OBJECT_NAME	VARCHAR	The name of the object on which you want to query HISTORICAL data. After running audit on a table, you can drop the table. In this case, object_name becomes NULL.
LICENSE_NAME	VARCHAR	The name of the license. After running a compliance audit, the value for this column is always <i>vertica</i> .
AUDIT_START_TIMESTAMP	TIMESTAMPTZ	When the audit started
AUDIT_END_TIMESTAMP	TIMESTAMPTZ	When the audit finished
CONFIDENCE_LEVEL_PERCENT	FLOAT	The confidence level of the size estimate
ERROR_TOLERANCE_PERCENT	FLOAT	The error tolerance used for the size estimate
USED_SAMPLING	BOOLEAN	Whether data was randomly sampled (if false, all of the data was analyzed)

Column Name	Data Type	Description
CONFIDENCE_INTERVAL_LOWER_BOUND_BYTES	INTEGER	The lower bound of the data size estimate within the confidence level
CONFIDENCE_INTERVAL_UPPER_BOUND_BYTES	INTEGER	The upper bound of the data size estimate within the confidence level
SAMPLE_COUNT	INTEGER	The number of data samples used to generate the estimate
CELL_COUNT	INTEGER	The number of cells in the database

USER_CLIENT_AUTH

Provides information about the client authentication methods that are associated with database users. You associate an authentication method with a user using [GRANT \(Authentication\)](#).

Column Name	Data Type	Description
USER_OID	INTEGER	A unique identifier for that user.
USER_NAME	VARCHAR	Name of the user.
AUTH_OID	INTEGER	A unique identifier for the authentication method you are using.
AUTH_NAME	VARCHAR	Name that you gave to the authentication method.
GRANTED_TO	BOOLEAN	Name of the user with whom you have associated the authentication method using GRANT (Authentication) .

USER_FUNCTION_PARAMETERS

Provides information about the parameters of a C++ user-defined function (UDx). You can only view parameters that have the `Properties.visible` parameter set to `TRUE`.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR (128)	The schema to which the function belongs.
FUNCTION_NAME	VARCHAR (128)	The name assigned by the user to the User-Defined Function.
FUNCTION_TYPE	VARCHAR (128)	The type of user-defined function. For example, 'User Defined Function'.
FUNCTION_ARGUMENT_TYPE	VARCHAR (8192)	The number and data types of input arguments for the function.
PARAMETER_NAME	VARCHAR (128)	The name of the parameter for the user-defined function.
DATA_TYPE	VARCHAR (128)	The data type of the parameter.
DATA_TYPE_ID	INTEGER	A number specifying the ID for the parameter's data type.
DATA_TYPE_LENGTH	INTEGER	The maximum length of the parameter's data type.
IS_REQUIRED	BOOLEAN	Indicates whether the parameter is required or not. If set to TRUE, and you don't provide the parameter, Vertica throws an error.
CAN_BE_NULL	BOOLEAN	Indicates whether the parameter can be passed as a NULL value. If set to FALSE, you pass the parameter with a NULL value, Vertica throws an error.
COMMENT	VARCHAR (128)	A user-supplied description of the parameter.

Privileges

Any user can query the USER_FUNCTION_PARAMETERS table. However, users can only see table information about those UDF functions which the user has permission to use.

See Also

- [Developing User-Defined Extensions \(UDxs\)](#)
- [UDx Parameters](#)

USER_FUNCTIONS

Returns metadata about user-defined SQL functions (which store commonly used SQL expressions as a function in the Vertica catalog) and User-Defined Functions (UDx).

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema in which this function exists.
FUNCTION_NAME	VARCHAR	The name assigned by the user to the SQL function or User-Defined Function.
PROCEDURE_TYPE	VARCHAR	The type of user-defined function. For example, 'User Defined Function'.
FUNCTION_RETURN_TYPE	VARCHAR	The data type name that the SQL function returns.
FUNCTION_ARGUMENT_TYPE	VARCHAR	The number and data types of parameters for the function.
FUNCTION_DEFINITION	VARCHAR	The SQL expression that the user defined in the SQL function's function body.
VOLATILITY	VARCHAR	The SQL function's volatility (whether a function returns the same output given the same input). Can be immutable, volatile, or stable.
IS_STRICT	BOOLEAN	Indicates whether the SQL function is strict, where <i>t</i> is true and <i>f</i> is false.
IS_FENCED	BOOLEAN	Indicates whether the function runs in Fenced Mode or not.
COMMENT	VARCHAR	A comment about this function provided by the function creator.

Notes

- The volatility and strictness of a SQL function are automatically inferred from the function definition in order that Vertica determine the correctness of usage, such as where an immutable function is expected but a volatile function is provided.
- The volatility and strictness of a UDX is defined by the UDX's developer.

Example

Create a SQL function called `myzeroifnull` in the public schema:

```
=> CREATE FUNCTION myzeroifnull(x INT) RETURN INT
  AS BEGIN
    RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
  END;
```

Now query the `USER_FUNCTIONS` table. The query returns just the `myzeroifnull` macro because it is the only one created in this schema:

```
=> SELECT * FROM user_functions;
-[ RECORD 1 ]-----+-----
schema_name      | public
function_name    | myzeroifnull
procedure_type   | User Defined Function
function_return_type | Integer
function_argument_type | x Integer
function_definition | RETURN CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END
volatility       | immutable
is_strict        | f
is_fenced        | f
comment          |
```

See Also

- [CREATE FUNCTION \(SQL Functions\)](#)
- [ALTER FUNCTION \(UDF or UDT\)](#)
- [DROP FUNCTION](#)

USER_PROCEDURES

Provides information about external procedures that have been defined for Vertica. User see only the procedures they can execute.

Column Name	Data Type	Description
PROCEDURE_NAME	VARCHAR	The name given to the external procedure through the CREATE PROCEDURE statement.
PROCEDURE_ARGUMENTS	VARCHAR	Lists arguments for the external procedure.
SCHEMA_NAME	VARCHAR	Indicates the schema in which the external procedure is defined.

Example

```
=> SELECT * FROM user_procedures;
  procedure_name | procedure_arguments | schema_name
-----+-----+-----
  helloplanet   | arg1 Varchar        | public
(1 row)
```

USER_TRANSFORMS

Lists the currently-defined user-defined transform functions (UDTFs).

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR (128)	The name of the schema containing the UDTF.
FUNCTION_NAME	VARCHAR (128)	The SQL function name assigned by the user.
FUNCTION_RETURN_TYPE	VARCHAR (128)	The data types of the columns the UDTF returns.
FUNCTION_ARGUMENT_	VARCHAR (8192)	The data types of the columns that make up the input row.

Column Name	Data Type	Description
TYPE		
FUNCTION_DEFINITION	VARCHAR (128)	A string containing the name of the factory class for the UDTF, and the name of the library that contains it.
IS_FENCED	BOOLEAN	Whether the UDTF runs in fenced mode.

Privileges

No explicit permissions are required; however, users see only UDTFs contained in schemas to which they have read access.

See Also

- [Transform Functions \(UDTFs\)](#)
- [CREATE TRANSFORM FUNCTION](#)

USERS

Provides information about all users in the database.

Tip: To see if a role has been assigned to a user, call the function [HAS_ROLE](#).

Column Name	Data Type	Description
USER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the user.
USER_NAME	VARCHAR	The user name for which information is listed.
IS_SUPER_USER	BOOLEAN	A system flag, where <i>t</i> (true) identifies the superuser created at the time of installation. All other users are denoted by <i>f</i> (false).
PROFILE_NAME	VARCHAR	The name of the profile to which the user is assigned. The profile controls the user's password policy.

Column Name	Data Type	Description
IS_LOCKED	BOOLEAN	Whether the user's account is locked. A locked user cannot log into the system.
LOCK_TIME	TIMESTAMPZ	When the user's account was locked. Used to determine when to automatically unlock the account, if the user's profile has a PASSWORD_LOCK_TIME parameter set.
RESOURCE_POOL	VARCHAR	The resource pool to which the user is assigned.
MEMORY_CAP_KB	VARCHAR	The maximum amount of memory a query run by the user can consume, in kilobytes.
TEMP_SPACE_CAP_KB	VARCHAR	The maximum amount of temporary disk space a query run by the user can consume, in kilobytes.
RUN_TIME_CAP	VARCHAR	The maximum amount of time any of the user's queries are allowed to run.
MAX_CONNECTIONS	VARCHAR	The maximum number of connections allowed for this user.
CONNECTION_LIMIT_MODE	VARCHAR	Indicates whether the user sets connection limits through the node or in database mode.
IDLE_SESSION_TIMEOUT	VARCHAR	The time the system waits before timing out the user's idle session. Maximum value is 1 year. See Interval Subtype Units for valid intervals.
GRACE_PERIOD	VARCHAR	Specifies how long a user query can block on any session socket, while awaiting client input or output. If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated.
ALL_ROLES	VARCHAR	Roles assigned to the user. An asterisk in ALL_ROLES output means role granted WITH ADMIN OPTION. See Database Roles in the Administrator's Guide.
DEFAULT_	VARCHAR	Default roles assigned to the user. An asterisk in

Column Name	Data Type	Description
ROLES		DEFAULT_ROLES output means role granted WITH ADMIN OPTION. See Default Roles for Database Users in the Administrator's Guide.
SEARCH_PATH	VARCHAR	Sets the default schema search path for the user. See Setting Search Paths in the Administrator's Guide.
LDAP_DN	VARCHAR	Indicates whether or not the Vertica Analytic Database user maps to an LDAP Link user. When the column is set to dn, the Vertica user maps to LDAP Link..
LDAP_URI_HASH	INTEGER	The URI hash number for the LDAP user.
IS_ORPHANED_FROM_LDAP	BOOLEAN	Indicates if the user is disconnected (orphaned) from LDAP, set to one of the following: <ul style="list-style-type: none">• t: User is orphaned• f : User is not orphaned For more information see Troubleshooting LDAP Link Issues

See Also

- [GRANTS](#)
- [HAS_ROLE](#)

VIEW_COLUMNS

Provides view attribute information.

Note: If you drop a table that is referenced by a view, Vertica does not drop the view. However, attempts to access information about it from VIEW_COLUMNS return an error that the view is invalid.

Column Name	Data Type	Description
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies this view.
TABLE_SCHEMA	VARCHAR	The name of this view's schema.
TABLE_NAME	VARCHAR	The view name.
COLUMN_ID	VARCHAR	A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in this view.
COLUMN_NAME	VARCHAR	The name of a column in this view.
DATA_TYPE	VARCHAR	The data type of a view column.
DATA_TYPE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies a view column's data type.
DATA_TYPE_LENGTH	INTEGER	The data type's maximum length.
CHARACTER_MAXIMUM_LENGTH	INTEGER	The column's maximum length, valid only for character types.
NUMERIC_PRECISION	INTEGER	The column's number of significant decimal digits.
NUMERIC_SCALE	INTEGER	The column's number of fractional digits.
DATETIME_PRECISION	INTEGER	For TIMESTAMP data type, returns the declared precision; returns null if no precision was declared.
INTERVAL_PRECISION	INTEGER	The number of fractional digits retained in the seconds field.
ORDINAL_POSITION	INTEGER	The position of the column relative to other columns in the view.

See Also

[VIEWS](#)

VIEWS

Provides information about all views within the system. See [Views](#) for more information.

Column Name	Data Type	Description
TABLE_SCHEMA_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the view schema.
TABLE_SCHEMA	VARCHAR	The name of the view schema.
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the view.
TABLE_NAME	VARCHAR	The view name.
OWNER_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the view owner.
OWNER_NAME	VARCHAR	View owner's user name
VIEW_DEFINITION	VARCHAR	The query that defines the view.
IS_SYSTEM_VIEW	BOOLEAN	Indicates whether the view is a system view.
SYSTEM_VIEW_CREATOR	VARCHAR	View creator's user name.
CREATE_TIME	TIMESTAMP	Specifies when this view was created.
IS_LOCAL_TEMP_VIEW	BOOLEAN	Indicates whether this view is a temporary view stored locally.
INHERIT_PRIVILEGES	BOOLEAN	Indicates whether inherited privileges are enabled for this view. For details, see Grant Inherited Privileges .

See Also

[VIEW_COLUMNS](#)

V_MONITOR Schema

The system tables in this section reside in the `v_monitor` schema. These tables provide information about the health of the Vertica database.

ACTIVE_EVENTS

Returns all active events in the cluster. See [Monitoring Events](#).

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name where the event occurred.
EVENT_CODE	INTEGER	A numeric ID that indicates the type of event. See Event Types for a list of event type codes.
EVENT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the specific event.
EVENT_SEVERITY	VARCHAR	The severity of the event from highest to lowest. These events are based on standard syslog severity types. <ul style="list-style-type: none">• 0—Emergency• 1—Alert• 2—Critical• 3—Error• 4—Warning• 5—Notice• 6—Informational• 7—Debug
EVENT_POSTED_TIMESTAMP	TIMESTAMP	The year, month, day, and time the event was reported. The time is posted in military time.

Column Name	Data Type	Description
EVENT_EXPIRATION	VARCHAR	The year, month, day, and time the event expire. The time is posted in military time. If the cause of the event is still active, the event is posted again.
EVENT_CODE_DESCRIPTION	VARCHAR	A brief description of the event and details pertinent to the specific situation.
EVENT_PROBLEM_DESCRIPTION	VARCHAR	A generic description of the event.
REPORTING_NODE	VARCHAR	The name of the node within the cluster that reported the event.
EVENT_SENT_TO_CHANNELS	VARCHAR	The event logging mechanisms that are configured for Vertica. These can include <code>vertica.log</code> , (configured by default) <code>syslog</code> , and <code>SNMP</code> .
EVENT_POSTED_COUNT	INTEGER	Tracks the number of times an event occurs. Rather than posting the same event multiple times, Vertica posts the event once and then counts the number of additional instances in which the event occurs.

ALLOCATOR_USAGE

Provides real-time information on the allocation and reuse of memory pools for a Vertica node.

There are two memory pools in Vertica, global and SAL. The global memory pool is related to Vertica catalog objects. The SAL memory pool is related to the system storage layer. These memory pools are physical structures from which Vertica allocates and reuses portions of memory.

Within the memory pools, there are two allocation types. Both global and SAL memory pools include chunk and object memory allocation types.

- *Chunk* allocations are from tiered storage, and are grouped into sizes, in bytes, that are powers of 2.
- *Object* allocations are object types, for example, a table or projection. Each object assumes a set size.

The table provides detailed information on these memory pool allocations.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node from which Vertica has collected this allocator information.
POOL_NAME	VARCHAR	One of two memory pools: <ul style="list-style-type: none"> • global: Memory pool is related to Vertica catalog objects. • SAL: Memory pool is related to the system storage layer.
ALLOCATION_TYPE	VARCHAR	One of two memory allocation types: <ul style="list-style-type: none"> • chunk: Chunk allocations are grouped into sizes that are powers of 2. • object: Object allocations assume a set amount of memory based upon the specific object.
UNIT_SIZE	INTEGER	The size, in bytes, of the memory allocation. For example, if the allocation type is a table (an object type), then Vertica allots 8 bytes.
FREE_COUNT	INTEGER	Indicates the count of blocks of freed memory that Vertica has reserved for future memory needs. For example, if you delete a table, Vertica reserves the 8 bytes originally allotted for the table. The 8 bytes freed become 1 unit of memory that Vertica adds to this column.
FREE_BYTES	INTEGER	Indicates the number of freed memory bytes. For example, with a table deletion, Vertica adds 8 bytes to this column. Note: Vertica does not release memory after originally allocating it, unless the node or database is restarted.
USED_COUNT	INTEGER	Indicates the count of in-use blocks for this

Column Name	Data Type	Description
		allocation. For example, if your database includes two table objects, Vertica adds 2 to this column.
USED_BYTES	INTEGER	The number of bytes of in-use blocks of memory. For example, if your database includes two table objects, each of which assume 8 bytes, Vertica adds 16 to this column.
TOTAL_SIZE	INTEGER	Indicates the number of bytes that is the sum of all free and used memory.
CAPTURE_TIME	TIMESTAMPTZ	Indicates the current timestamp for when Vertica collected the for this table.
ALLOCATION_NAME	VARCHAR	Provides the name of the allocation type. <ul style="list-style-type: none"> If the allocation is an object type, provides the name of the object. For example, CAT : :Schema. Object types can also have the name internal, meaning that the object is an internal data structure. <p>Those object types that are not internal are prefaced with either CAT or SAL. Those prefaced with CAT indicate memory from the global memory pool. SAL indicates memory from the system storage memory pool.</p> <ul style="list-style-type: none"> If the allocation type is chunk, indicates a power of 2 in this field to represent the number of bytes assumed by the chunk. For example, 2^5.

Sample: How Memory Pool Memory is Allotted, Retained, and Freed

The following table shows sample column values based upon a hypothetical example. The sample illustrates how column values change based upon addition or deletion of a table object.

- When you add a table object (t1), Vertica assumes a UNIT_SIZE of 8 bytes, with a USED_COUNT of 1.
- When you add a second table object (t2), the USED_COUNT increases to 2. Since each object assumes 8 bytes, USED_BYTES increases to 16.
- When you delete one of the two table objects, Vertica USED_COUNT decreases to 1, and USED_BYTES decreases to 8. Since Vertica retains the memory for future use, FREE_BYTES increases to 8, and FREE_COUNT increases to 1.
- Finally, when you create a new table object (t3), Vertica frees the memory for reuse. FREE_COUNT and FREE_BYTES return to 0.

Column Names	Add One Table Object (t1)	Add a Second Table Object (t2)	Delete a Table Object (t2)	Create a New Table Object (t3)
NODE_NAME	v_vmart_node0001	v_vmart_node0001	v_vmart_node0001	v_vmart_node0001
POOL_NAME	global	global	global	global
ALLOCATION_TYPE	object	object	object	object
UNIT_SIZE	8	8	8	8
FREE_COUNT	0	0	1	0
FREE_BYTES	0	0	8	0
USED_COUNT	1	2	1	2
USED_BYTES	8	16	8	16
TOTAL_SIZE	8	16	16	16
CAPTURE_TIME	2017-05-24 13:28:07.838 55-04	2017-05-24 14:16:04.4809 53-04	2017-05-24 14:16:32.0773 22-04	2017-05-24 14:17:07.3207 45-04
ALLOCATION_NAME	CAT::Table	CAT::Table	CAT::Table	CAT::Table

Example

The following example shows one sample record for a chunk allocation type, and one for an object type.

```
=> \x
Expanded display is on.

=> select * from allocator_usage;
-[ RECORD 1 ]-----+-----
node_name      | v_vmart_node0004
pool_name      | global
allocation_type| chunk
unit_size     | 8
free_count     | 1069
free_bytes     | 8552
used_count     | 7327
used_bytes     | 58616
total_size     | 67168
capture_time   | 2017-05-24 13:28:07.83855-04
allocation_name| 2^3
.
.
.
-[ RECORD 105 ]--+-----
node_name      | v_vmart_node0004
pool_name      | SAL
allocation_type| object
unit_size     | 128
free_count     | 0
free_bytes     | 0
used_count     | 2
used_bytes     | 256
total_size     | 256
capture_time   | 2017-05-24 14:44:30.153892-04
allocation_name| SAL::WOSAAlloc
.
.
.
```

COLUMN_STORAGE

Returns the amount of disk storage used by each column of each projection on each node. WOS data is stored by row, so per-column byte counts are not available.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
COLUMN_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the column.
COLUMN_NAME	VARCHAR	The column name for which information is listed.
ROW_COUNT	INTEGER	The number of rows in the column.
USED_BYTES	INTEGER	The disk storage allocation of the column in bytes.
ENCODINGS	VARCHAR	The encoding type for the column.
COMPRESSION	VARCHAR	The compression type for the column. You can compare ENCODINGS and COMPRESSION columns to see how different encoding types affect column storage when optimizing for compression.
WOS_ROW_COUNT	INTEGER	The number of WOS rows in the column.
ROS_ROW_COUNT	INTEGER	The number of ROS rows in the column.
ROS_USED_BYTES	INTEGER	The number of ROS bytes in the column.
ROS_COUNT	INTEGER	The number of ROS containers.
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	The associated projection name for the column.
PROJECTION_SCHEMA	VARCHAR	The name of the schema associated with the projection.
ANCHOR_TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the anchor table.
ANCHOR_TABLE_NAME	VARCHAR	The associated table name.
ANCHOR_TABLE_SCHEMA	VARCHAR	The associated table's schema name.
ANCHOR_TABLE_COLUMN_ID	VARCHAR	A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.
ANCHOR_TABLE_COLUMN_NAME	VARCHAR	The name of the anchor table.

CONFIGURATION_CHANGES

Records the change history of system [configuration parameters](#). This information is useful for identifying:

- Who changed the configuration parameter value
- When the configuration parameter was changed
- Whether nonstandard settings were in effect in the past

Column Name	Data Type	Description
EVENT_TIMESTAMP	TIMESTAMPTZ	Time when the row was recorded.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_ID	INTEGER	Identifier of the user who changed configuration parameters.
USER_NAME	VARCHAR	Name of the user who changed configuration parameters at the time Vertica recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
PARAMETER	VARCHAR	Name of the changed parameter. See Configuration Parameters in the Administrator's Guide for a detailed list of supported parameters.
VALUE	VARCHAR	New value of the configuration parameter.

Privileges

Superuser

CONFIGURATION_PARAMETERS

Provides information about configuration parameters currently in use by the system that are configurable at the database, node, or session level.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node names on the cluster for which information is listed. ALL indicates that all the nodes have the same value.
PARAMETER_NAME	VARCHAR	The name of the configurable parameter. For names of supported parameters, see Configuration Parameter Categories in the Administrator's Guide.
CURRENT_VALUE	VARCHAR	The value of the current setting for the parameter.
RESTART_VALUE	VARCHAR	The value of the parameter after the next restart.
DATABASE_VALUE	VARCHAR	The value that is set at the database level. If no database-level value is set, the value reflects the default value.
DEFAULT_VALUE	VARCHAR	The default value for the parameter.
CURRENT_LEVEL	VARCHAR	Level at which CURRENT_VALUE is set. Valid values: Node, database, session, or default.
RESTART_LEVEL	VARCHAR	Level at which the parameter will be set after the next restart. Valid values: Node, database, or default.
IS_MISMATCH	BOOLEAN	A <i>t</i> (true) setting indicates CURRENT_VALUE and RESTART_VALUE do not match.
GROUPS	VARCHAR	Any group to which the parameter belongs (for

Column Name	Data Type	Description
		example, Security Parameters).
ALLOWED_LEVELS	VARCHAR	Indicates level or levels at which the specified parameter can be set. Valid values: Node, database, or session.
SUPERUSER_ONLY	BOOLEAN	Indicates if the parameter settings are viewable by the superuser only. If true, the following columns will be masked if viewed by a non-superuser: <ul style="list-style-type: none"> • current_value • restart_value • database_value • default_value
CHANGE_UNDER_SUPPORT_GUIDANCE	BOOLEAN	A <i>t</i> (true) setting indicates parameters intended for use only by Vertica.
CHANGE_REQUIRES_RESTART	BOOLEAN	Indicates whether the configuration change requires a restart, where <i>t</i> is true and <i>f</i> is false.
DESCRIPTION	VARCHAR	A description of the parameter's purpose.

Non-Default Locales Error

The CONFIGURATION_PARAMETERS table returns the following error in non-default locales:

```
ERROR: ORDER BY is not supported with UNION/INTERSECT/EXCEPT in non-default locales
HINT: Please move the UNION to a FROM clause subquery.
```

See the [SET LOCALE](#) command for details.

Example

The following example shows a case where the parameter requires a restart for the new setting to take effect:

```
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'EnableSSL';
-[ RECORD 1 ]-----+-----
node_name           | ALL
parameter_name     | EnableSSL
current_value       | 0
restart_value       | 1
database_value      | 0
default_value       | 0
current_level       | DEFAULT
restart_level       | NODE
is_mismatch         | t
groups              |
allowed_levels      | NODE, DATABASE
superuser_only      | f
change_under_support_guidance | f
change_requires_restart | t
description         | Enable SSL for the server
```

The following example shows a case where a non-superuser is viewing a parameter where `superuser_only` is true.

```
=> \c VMart nonSuperuser
You are now connected to database "VMart" as user "nonSuperuser".
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE where superuser_only = 't';
-[ RECORD 1 ]-----+-----
node_name           | ALL
parameter_name     | LDAPLinkDryRun
current_value       | *****
restart_value       | *****
database_value      | *****
default_value       | *****
current_level       | DEFAULT
restart_level       | DEFAULT
is_mismatch         | f
groups              |
allowed_levels      | DATABASE
superuser_only      | t
change_under_support_guidance | t
change_requires_restart | f
description         | Just contact LDAP server and log the response. Don't perform any
changes
```

See Also

[Configuration Parameters](#) in the Administrator's Guide

CPU_USAGE

Records CPU usage history on the system.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
START_TIME	TIMESTAMP	Beginning of history interval.
END_TIME	TIMESTAMP	End of history interval.
AVERAGE_CPU_USAGE_PERCENT	FLOAT	Average CPU usage in percent of total CPU time (0-100) during history interval.

Privileges

Superuser

CRITICAL_HOSTS

Lists the critical hosts whose failure would cause the database to become unsafe and force a shutdown.

Column Name	Data Type	Description
HOST_NAME	VARCHAR	Name of a critical host

Privileges

None

CRITICAL_NODES

Lists the critical nodes whose failure would cause the database to become unsafe and force a shutdown.

Column Name	Data Type	Description
NODE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the node.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of a critical node.

CURRENT_SESSION

Returns information about the current active session. Use this table to find out the current session's `sessionID` and get the duration of the previously-run query.

Column Name	Data Type	Description																												
NODE_NAME	VARCHAR	Name of the node for which information is listed																												
USER_NAME	VARCHAR	Name used to log into the database, NULL if the session is internal																												
CLIENT_HOSTNAME	VARCHAR	Host name and port of the TCP socket from which the client connection was made, NULL if the session is internal																												
TYPE	INTEGER	Identifies the session type, one of the following integer values: <table border="1" data-bbox="797 1207 1409 1780"> <tbody> <tr> <td>1</td> <td>Client</td> <td>8</td> <td>Shutdown</td> </tr> <tr> <td>2</td> <td>DBD</td> <td>9</td> <td>License audit</td> </tr> <tr> <td>3</td> <td>Merge out</td> <td>10</td> <td>Timer service</td> </tr> <tr> <td>4</td> <td>Move out</td> <td>11</td> <td>Connection</td> </tr> <tr> <td>5</td> <td>Rebalance cluster</td> <td>12</td> <td>VSpread</td> </tr> <tr> <td>6</td> <td>Recovery</td> <td>13</td> <td>Sub-session</td> </tr> <tr> <td>7</td> <td>Refresh</td> <td>14</td> <td>Repartition table</td> </tr> </tbody> </table>	1	Client	8	Shutdown	2	DBD	9	License audit	3	Merge out	10	Timer service	4	Move out	11	Connection	5	Rebalance cluster	12	VSpread	6	Recovery	13	Sub-session	7	Refresh	14	Repartition table
1	Client	8	Shutdown																											
2	DBD	9	License audit																											
3	Merge out	10	Timer service																											
4	Move out	11	Connection																											
5	Rebalance cluster	12	VSpread																											
6	Recovery	13	Sub-session																											
7	Refresh	14	Repartition table																											
CLIENT_PID	INTEGER	Process identifier of the client process that issued this connection. This process might be																												

Column Name	Data Type	Description
		on a different machine than the server.
LOGIN_TIMESTAMP	TIMESTAMP	When the user logged into the database or the internal session was created. This column can help identify open sessions that are idle.
SESSION_ID	VARCHAR	Identifier required to close or interrupt a session. This identifier is unique within the cluster at any point in time, but can be reused when the session closes.
CLIENT_LABEL	VARCHAR	User-specified label for the client connection that can be set when using ODBC. See Label in Data Source Name (DSN) Connection Properties in Connecting to Vertica.
TRANSACTION_START	TIMESTAMP	When the current transaction started, NULL if no transaction is running
TRANSACTION_ID	VARCHAR	Hexadecimal identifier of the current transaction, NULL if no transaction is in progress
TRANSACTION_DESCRIPTION	VARCHAR	Description of the current transaction
STATEMENT_START	TIMESTAMP	When the current statement started execution, NULL if no statement is running
STATEMENT_ID	VARCHAR	Unique numeric ID for the currently-running statement, NULL if no statement is being processed. Combined, TRANSACTION_ID and STATEMENT_ID uniquely identify a statement within a session.
LAST_STATEMENT_DURATION_US	INTEGER	Duration in microseconds of the last completed statement
CURRENT_STATEMENT	VARCHAR	The currently-running statement, if any. NULL indicates that no statement is currently being processed.

Column Name	Data Type	Description
LAST_STATEMENT	VARCHAR	NULL if the user has just logged in, otherwise the currently running statement or most recently completed statement.
EXECUTION_ENGINE_PROFILING_CONFIGURATION	VARCHAR	See Profiling Settings below.
QUERY_PROFILING_CONFIGURATION	VARCHAR	See Profiling Settings below.
SESSION_PROFILING_CONFIGURATION	VARCHAR	See Profiling Settings below.
CLIENT_TYPE	VARCHAR	Type of client from which the connection was made, one of the following: <ul style="list-style-type: none"> • ADO.NET Driver • ODBC Driver • JDBC Driver • vsql
CLIENT_VERSION	VARCHAR	Client version
CLIENT_OS	VARCHAR	Client operating system
CLIENT_OS_USER_NAME	VARCHAR	Identifies the user that logged into the database, also set for unsuccessful login attempts.
REQUESTED_PROTOCOL	VARCHAR	Communication protocol version that the ODBC client driver sends to Vertica server, used to support backward compatibility with earlier server versions.
EFFECTIVE_PROTOCOL	VARCHAR	Minimum protocol version supported by client and driver.

Profiling Settings

The following columns show settings for different [profiling categories](#):

- EXECUTION_ENGINE_PROFILING_CONFIGURATION
- QUERY_PROFILING_CONFIGURATION
- SESSION_PROFILING_CONFIGURATION

These are set to one of the following:

Empty	No profiling is set
Session	On for current session
Global	On by default for all sessions
Session, Global	On by default for all sessions, including current session.

For information about controlling profiling settings, see [Enabling and Disabling Profiling](#) in the Administrator's Guide.

DATA_COLLECTOR

Shows the Data Collector components, their current retention policies, and statistics about how much data is retained and how much has been discarded for various reasons. DATA_COLLECTOR also calculates approximate collection rate, to aid in sizing calculations.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name on which information is retained.
COMPONENT	VARCHAR	The name of the component and its policy.
TABLE_NAME	VARCHAR	The data collector (dc) table name for which information is listed.
DESCRIPTION	VARCHAR	A short description about the component.
IN_DB_LOG	BOOLEAN	Denotes if monitoring information is retained in the dbLog file.

Column Name	Data Type	Description
IN_VERTICA_LOG	BOOLEAN	Denotes if monitoring information is retained in the <code>vertica.log</code> file.
MEMORY_BUFFER_SIZE_KB	INTEGER	The size of the memory buffer in kilobytes.
DISK_SIZE_KB	INTEGER	The on-disk size of the table in kilobytes.
SET_INTERVAL	BOOLEAN	A <code>t</code> (true) setting indicates time-based retention is set.
INTERVAL_TIME_S	INTERVAL	The time of retention expressed as an <code>INTERVAL</code> type. To turn time-based retention off, set the value to 0.
RECORD_TOO_BIG_ERRORS	INTEGER	A number that increments by one each time an error is thrown because data did not fit in memory (based on the data collector retention policy).
LOST_BUFFERS	INTEGER	The number of buffers lost.
LOST_RECORDS	INTEGER	The number of records lost.
RETIRED_FILES	INTEGER	The number of retired files.
RETIRED_RECORDS	INTEGER	The number of retired records.
CURRENT_MEMORY_RECORDS	INTEGER	The current number of rows in memory.
CURRENT_DISK_RECORDS	INTEGER	The current number of rows stored on disk.
CURRENT_MEMORY_BYTES	INTEGER	Total current memory used in kilobytes.
CURRENT_DISK_BYTES	INTEGER	Total current disk space used in kilobytes.
FIRST_TIME	TIMESTAMP	Timestamp of the first record.
LAST_TIME	TIMESTAMP	Timestamp of the last record
KB_PER_DAY	FLOAT	Total kilobytes used per day.

Notes

- Data Collector is on by default. To turn it off, see [Enabling and Disabling Data Collector](#)
- You can configure monitoring information retention policies. See [Data Collector Functions](#) and [Configuring Data Retention Policies](#) for more information.

Examples

The following example shows how to return all component names and their descriptions. This is a useful query if you want to change the retention policy for a particular component and don't remember its name:

```
=> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;
```

component	description
AllocationPoolStatistics	Information about global memory pools ...
AllocationPoolStatisticsByDay	Information about global memory pools, ... (historical, by day)
AllocationPoolStatisticsByHour	Information about global memory pools, ... (historical, by hour)
AllocationPoolStatisticsByMinute	Information about global memory pools, ... (historical, by minute)
AllocationPoolStatisticsBySecond	Information about global memory pools, ... (historical, by second)
AnalyzeStatistics	History of statistics collection
Backups	Monitoring successful backups
CatalogInfo	Catalog statistics and history
CatalogInfoByDay	Catalog statistics and history (historical, by day)
CatalogInfoByHour	Catalog statistics and history (historical, by hour)
CatalogInfoByMinute	Catalog statistics and history (historical, by minute)
CatalogInfoBySecond	Catalog statistics and history (historical, by second)
ClientServerMessages	Client-Server Messages (Front End to Back End Protocol) sent
ConfigurationChanges	Changes to configuration parameters (vertica.conf)
CpuAggregate	Aggregate CPU information
CpuAggregateByDay	Aggregate CPU information (historical, by day)
CpuAggregateByHour	Aggregate CPU information (historical, by hour)
CpuAggregateByMinute	Aggregate CPU information (historical, by minute)
CpuAggregateBySecond	Aggregate CPU information (historical, by second)
CpuInfo	CPU information
CpuInfoByDay	CPU information (historical, by day)
CpuInfoByHour	CPU information (historical, by hour)
CpuInfoByMinute	CPU information (historical, by minute)
CpuInfoBySecond	CPU information (historical, by second)
DeploymentsCompleted	History of designs deployed
DesignsCompleted	History of designs executed
DiskResourceRejections	Disk Resource Rejection Records
Errors	History of all errors+warnings encountered
ExecutionEngineEvents	History of important events during local planning and execution
ExecutionEngineProfiles	History of EE profiles
.	.
.	.

(93 rows)

Related Topics

- [Data Collector Functions](#)
- [Retaining Monitoring Information](#) and [How Vertica Calculates Database Size](#) in the Administrator's Guide
- [SET_DATA_COLLECTOR_TIME_POLICY](#)
- [SET_DATA_COLLECTOR_POLICY](#)

DATABASE_BACKUPS

Lists historical information for each backup that successfully completed after running the `vbr` utility. This information is useful for determining whether to create a new backup before you advance the AHM. Because this system table displays historical information, its contents do not always reflect the current state of a backup repository. For example, if you delete a backup from a repository, the `DATABASE_BACKUPS` system table continues to display information about it.

To list existing backups, run `vbr` as described in [Viewing Backups](#) in the Administrator's Guide.

Column Name	Data Type	Description
BACKUP_TIMESTAMP	TIMESTAMP	The timestamp of the backup.
NODE_NAME	VARCHAR	The name of the initiator node that performed the backup logging.
SNAPSHOT_NAME	VARCHAR	The name of the backup, as specified in the <code>snapshotName</code> parameter of the <code>vbr</code> configuration file.
BACKUP_EPOCH	INTEGER	The database epoch at which the backup was saved.
NODE_COUNT	INTEGER	The number of nodes backed up in the completed backup, and as listed in the <code>[Mapping<i>n</i>]</code> sections of the configuration file.
OBJECTS	VARCHAR	The name of the object(s) contained in an object-level

Column Name	Data Type	Description
		backup. This column is empty if the record is for a full cluster backup.
FILE_SYSTEM_TYPE	VARCHAR	The type of file system, such as Linux.

Privileges

Superuser

DATABASE_CONNECTIONS

Lists the connections to other databases for importing and exporting data. See [Importing and Exporting Data Across Databases](#) in the Administrator's Guide.

Column Name	Data Type	Description
DATABASE	VARCHAR	The name of the connected database
USERNAME	VARCHAR	The username used to create the connection
HOST	VARCHAR	The host name used to create the connection
PORT	VARCHAR	The port number used to create the connection
ISVALID	BOOLEAN	Whether the connection is still open and usable or not

Example

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON '10.10.20.150',5433;
CONNECT
=> SELECT * FROM DATABASE_CONNECTIONS;
  database | username |      host      | port | isvalid
-----+-----+-----+-----+-----
  vmart    | dbadmin  | 10.10.20.150  | 5433 | t
(1 row)
```

DELETE_VECTORS

Holds information on deleted rows to speed up the delete process.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node storing the deleted rows.
SCHEMA_NAME	VARCHAR	The name of the schema where the deleted rows are located.
PROJECTION_NAME	VARCHAR	The name of the projection where the deleted rows are located.
STORAGE_TYPE	VARCHAR	The type of storage containing the delete vector (WOS or ROS).
DV_OID	INTEGER	The unique numeric ID (OID) that identifies this delete vector.
STORAGE_OID	INTEGER	The unique numeric ID (OID) that identifies the storage container that holds the delete vector.
SAL_STORAGE_ID	VARCHAR	Unique hexadecimal numeric ID assigned by the Vertica catalog, which identifies the storage.
DELETED_ROW_COUNT	INTEGER	The number of rows deleted.
USED_BYTES	INTEGER	The number of bytes used to store the deletion.
START_EPOCH	INTEGER	The start epoch of the data in the delete vector.
END_EPOCH	INTEGER	The end epoch of the data in the delete vector.
IS_SORTED	BOOLEAN	Whether the storage container's data is sorted (WOS containers only).

DEPLOY_STATUS

Records the history of deployed Database Designer designs and their deployment steps.

Column Name	Data Type	Description
EVENT_TIME	TIMESTAMP	Time when the row recorded the event.
USER_NAME	VARCHAR	Name of the user who deployed a design at the time Vertica recorded the session.

Column Name	Data Type	Description
DEPLOY_NAME	VARCHAR	Name the deployment, same as the user-specified design name.
DEPLOY_STEP	VARCHAR	Steps in the design deployment.
DEPLOY_STEP_STATUS	VARCHAR	Textual status description of the current step in the deploy process.
DEPLOY_STEP_COMPLETE_PERCENT	FLOAT	Progress of current step in percentage (0–100).
DEPLOY_COMPLETE_PERCENT	FLOAT	Progress of overall deployment in percentage (0–100).
ERROR_MESSAGE	VARCHAR	Error or warning message during deployment.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

DEPLOYMENT_PROJECTION_STATEMENTS

Contains information about [CREATE PROJECTION](#) statements used to deploy a database design. Each row contains information about a different [CREATE PROJECTION](#) statement. The function [DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY](#) populates this table.

Column Name	Column Type	Description
DEPLOYMENT_ID	INTEGER	Unique ID that Database Designer assigned to the deployment.
DESIGN_NAME	VARCHAR	Unique name that the user assigned to the design.
DEPLOYMENT_PROJECTION_ID	INTEGER	Unique ID assigned to the output projection by Database Designer.
STATEMENT_ID	INTEGER	Unique ID assigned to the statement type that creates the projection.

Column Name	Column Type	Description
STATEMENT	VARCHAR	Text for the statement that creates the projection.

DEPLOYMENT_PROJECTIONS

Contains information about projections created and dropped during the design. Each row contains information about a different projection. Database Designer populates this table after the design is deployed.

Column Name	Column Type	Description
deployment_id	INTEGER	Unique ID that Database Designer assigned to the deployment.
deployment_projection_id	INTEGER	Unique ID that Database Designer assigned to the output projection.
design_name	VARCHAR	Name of the design being deployed.
deployment_projection_name	VARCHAR	Name that Database Designer assigned to the projection.
anchor_table_schema	VARCHAR	Name of the schema that contains the table the projection is based on.
anchor_table_name	VARCHAR	Name of the table the projection is based on.
deployment_operation	VARCHAR	Action being taken on the projection, for example, add or drop.
deployment_projection_type	VARCHAR	Indicates whether Database Designer has proposed new projections for this design (DBD) or is using the existing catalog design (CATALOG). The REENCODED suffix indicates that the projection sort order and segmentation are the same, but the projection columns have new encodings: <ul style="list-style-type: none"> • DBD

Column Name	Column Type	Description
		<ul style="list-style-type: none"> CATALOG DBD_REENCODED CATALOG_REENCODED
deploy_weight	INTEGER	Weight of this projection in creating the design. This field is always 0 for projections that have been dropped.
estimated_size_on_disk	INTEGER	Approximate size of the projection on disk, in MB.

DESIGN_QUERIES

Contains info about design queries for a given design. The following functions populate this table:

- [DESIGNER_ADD_DESIGN_QUERIES](#)
- [DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS](#)
- [DESIGNER_ADD_DESIGN_QUERY](#)

Column Name	Column Type	Description
design_id	INTEGER	Unique id that Database Designer assigned to the design.
design_name	VARCHAR	Name that you specified for the design.
design_query_id	INTEGER	Unique id that Database Designer assigned to the design query.
design_query_id_index	INTEGER	Database Designer chunks the query text if it exceeds the maximum attribute size before storing it in this table. Database Designer stored all chunks stored under the same value of DESIGN_QUERY_ID. DESIGN_QUERY_ID_INDEX keeps track of the order of the chunks, starting with 0 and ending in n, the index of the final chunk.

Column Name	Column Type	Description
query_text	VARCHAR	Text of the query chunk, or the entire query text if it does not exceed the maximum attribute size.
weight	FLOAT	A value from 0 to 1 that indicates the importance of that query in creating the design. Assign a higher weight to queries that you run frequently so that Database Designer prioritizes those queries in creating the design. Default: 1.
design_query_search_path	VARCHAR	The search path with which the query is to be parsed.
design_query_signature	INTEGER	An INTEGER that categorizes queries that affect the design that Database Designer creates in the same way. Database Designer assigns a signature to each query, weights one query for each signature group, depending on how many queries there are with that signature, and Database Designer considers that query when creating the design.

Example

Add queries to VMART_DESIGN and query the DESIGN_QUERIES table:

```
=> SELECT DESIGNER_ADD_DESIGN_QUERIES('VMART_DESIGN', '/tmp/examples/vmart_queries.sql','true');
DESIGNER_ADD_DESIGN_QUERIES
-----
Number of accepted queries                =9
Number of queries referencing non-design tables =0
Number of unsupported queries             =0
Number of illegal queries                 =0
=> \x
Expanded display is on.
=> SELECT * FROM V_MONITOR.DESIGN.QUERIES
-[ RECORD 1 ]-----+-----
design_id          | 45035996273705090
design_name        | vmart_design
design_query_id    | 1
design_query_id_index | 0
query_text        | SELECT fat_content
FROM (
SELECT DISTINCT fat_content
FROM product_dimension
WHERE department_description
```

```

    IN ('Dairy') ) AS food
    ORDER BY fat_content
    LIMIT 5;
weight                | 1
design_query_search_path | v_dbd_vmart_design_vmart_design_ltt, "$user", public, v_catalog, v_
monitor, v_internal
design_query_signature  | 45035996273724651

-[ RECORD 2]-----+-----
design_query_id        | 2
design_query_id_index  | 0
query_text            | SELECT order_number, date_ordered
                      | FROM store.store_orders_fact orders
                      | WHERE orders.store_key IN (
                      | SELECT store_key
                      | FROM store.store_dimension
                      | WHERE store_state = 'MA')
                      | AND orders.vendor_key NOT IN (
                      | SELECT vendor_key
                      | FROM public.vendor_dimension
                      | WHERE vendor_state = 'MA')
                      | AND date_ordered < '2012-03-01';

weight                | 1
design_query_search_path | v_dbd_vmart_design_vmart_design_ltt, "$user", public, v_catalog, v_
monitor, v_internal
design_query_signature  | 45035996273724508
-[ RECORD 3]-----+-----
...

```

DESIGN_STATUS

Records the progress of a running Database Designer design or history of the last Database Designer design executed by the current user.

Column Name	Data Type	Description
event_time	TIMESTAMP	Time when the row recorded the event.
user_name	VARCHAR	Name of the user who ran a design at the time Vertica recorded the session.
design_name	VARCHAR	Name of the user-specified design.
design_phase	VARCHAR	Phase of the design.
phase_step	VARCHAR	Substep in each design phase
phase_step_complete_percent	FLOAT	Progress of current substep in percentage (0–100).

Column Name	Data Type	Description
phase_complete_percent	FLOAT	Progress of current design phase in percentage (0–100).

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Example

The following example shows the content of the DESIGN_STATUS table of a complete Database Designer run:

```
=> SELECT event_time, design_name, design_phase, phase_complete_percent
      FROM v_monitor.design_status;
event_time          | design_name | design_phase                               | phase_complete_
percent
-----+-----+-----+-----
-----
2012-02-14 10:31:20 | design1     | Design started                             |
2012-02-14 10:31:21 | design1     | Design in progress: Analyze statistics phase |
2012-02-14 10:31:21 | design1     | Analyzing data statistics                   | 33.33
2012-02-14 10:31:22 | design1     | Analyzing data statistics                   | 66.67
2012-02-14 10:31:24 | design1     | Analyzing data statistics                   | 100
2012-02-14 10:31:25 | design1     | Design in progress: Query optimization phase |
2012-02-14 10:31:25 | design1     | Optimizing query performance                | 37.5
2012-02-14 10:31:31 | design1     | Optimizing query performance                | 62.5
2012-02-14 10:31:36 | design1     | Optimizing query performance                | 75
2012-02-14 10:31:39 | design1     | Optimizing query performance                | 87.5
2012-02-14 10:31:41 | design1     | Optimizing query performance                | 87.5
2012-02-14 10:31:42 | design1     | Design in progress: Storage optimization phase |
2012-02-14 10:31:44 | design1     | Optimizing storage footprint                | 4.17
2012-02-14 10:31:44 | design1     | Optimizing storage footprint                | 16.67
2012-02-14 10:32:04 | design1     | Optimizing storage footprint                | 29.17
2012-02-14 10:32:04 | design1     | Optimizing storage footprint                | 31.25
2012-02-14 10:32:05 | design1     | Optimizing storage footprint                | 33.33
2012-02-14 10:32:05 | design1     | Optimizing storage footprint                | 35.42
2012-02-14 10:32:05 | design1     | Optimizing storage footprint                | 37.5
2012-02-14 10:32:05 | design1     | Optimizing storage footprint                | 62.5
2012-02-14 10:32:39 | design1     | Optimizing storage footprint                | 87.5
2012-02-14 10:32:39 | design1     | Optimizing storage footprint                | 87.5
2012-02-14 10:32:41 | design1     | Optimizing storage footprint                | 100
2012-02-14 10:33:12 | design1     | Design completed successfully               |
(24 rows)
```

DESIGN_TABLES

Contains information about all the design tables for all the designs for which you are the owner. Each row contains information about a different design table. Vertica creates this table when you run [DESIGNER_CREATE_DESIGN](#).

Column Name	Column Type	Description
design_name	VARCHAR	Unique name that the user specified for the design.
design_table_id	INTEGER	Unique ID that Database Designer assigned to the design table.
table_schema	VARCHAR	Name of the schema that contains the design table.
table_id	INTEGER	System object identifier (OID) assigned to the design table.
table_name	VARCHAR	Name of the design table.

Example

Add all the tables from the VMart database to the design VMART_DESIGN. This operation populates the DESIGN_TABLES table:

```
=> SELECT DESIGNER_ADD_DESIGN_TABLES('VMART_DESIGN','online_sales.*');
DESIGNER_ADD_DESIGN_TABLES
-----
3
(1 row)
=> SELECT DESIGNER_ADD_DESIGN_TABLES('VMART_DESIGN','public.*');
DESIGNER_ADD_DESIGN_TABLES
-----
9
(1 row)
=> SELECT DESIGNER_ADD_DESIGN_TABLES('VMART_DESIGN','store.*');
DESIGNER_ADD_DESIGN_TABLES
-----
3
(1 row)
=> SELECT * FROM DESIGN_TABLES;
design_name | design_table_id | table_schema | table_id | table_name
-----+-----+-----+-----+-----
VMART_DESIGN | 1 | online_sales | 45035996373718754 | online_page_dimension
VMART_DESIGN | 2 | online_sales | 45035996373718758 | call_center_dimension
```

```

VMART_DESIGN |          3 | online_sales | 45035996373718762 | online_sales_fact
VMART_DESIGN |          4 | public      | 45035996373718766 | customer_dimension
VMART_DESIGN |          5 | public      | 45035996373718770 | product_dimension
VMART_DESIGN |          6 | public      | 45035996373718774 | promotion_dimension
VMART_DESIGN |          7 | public      | 45035996373718778 | date_dimension
VMART_DESIGN |          8 | public      | 45035996373718782 | vendor_dimension
VMART_DESIGN |          9 | public      | 45035996373718786 | employee_dimension
VMART_DESIGN |         10 | public      | 45035996373718822 | shipping_dimension
VMART_DESIGN |         11 | public      | 45035996373718826 | warehouse_dimension
VMART_DESIGN |         12 | public      | 45035996373718830 | inventory_face
VMART_DESIGN |         13 | store       | 45035996373718794 | store_dimension
VMART_DESIGN |         14 | store       | 45035996373718798 | store_sales_fact
VMART_DESIGN |         15 | store       | 45035996373718812 | store_orders_fact
(15 rows)

```

DESIGNS

Contains information about a Database Designer design. After you create a design and specify certain parameters for Database Designer, [DESIGNER_CREATE_DESIGN](#) creates this table in the V_MONITOR schema.

Column Name	Column Type	Description
design_id	INTEGER	Unique ID that Database Designer assigns to this design.
design_name	VARCHAR	Name that the user specifies for the design.
ksafety_level	INTEGER	K-safety level for the design. Database Designer assigns a K-safety value of 0 for clusters with 1 or 2 nodes, and assigns a value of 1 for clusters with 3 or more nodes.
optimization_objective	VARCHAR	Name of the optimization objective for the design. Valid values are: <ul style="list-style-type: none"> • QUERY • LOAD • BALANCED (default)
design_type	VARCHAR	Name of the design type. Valid values are: <ul style="list-style-type: none"> • COMPREHENSIVE (default) • INCREMENTAL

Column Name	Column Type	Description
propose_super_first	BOOLEAN	Specifies to propose superprojections before projections, by default f. If DESIGN_MODE is COMPREHENSIVE, this field has no impact.
design_available	BOOLEAN	t if the design is currently available, otherwise, f (default).
collected_statistics	BOOLEAN	t if statistics are to be collected when creating the design, otherwise, f (default).
populate_design_tables_from_queries	BOOLEAN	t if you want to populate the design tables from the design queries, otherwise, f (default).
encoding_design	BOOLEAN	t if the design is an encoding optimization design on pre-existing projections, otherwise, f (default).
deployment_parallelism	INTEGER	Number of tables to be deployed in parallel when the design is complete. Default: 0
propose_unsegmented_projections	BOOLEAN	t if you specify unsegmented projections, otherwise, f (default).
analyze_correlations_mode	INTEGER	<p>Specifies how Database Designer should handle existing column correlations in a design and whether or not Database Designer should reanalyze existing column correlations.</p> <ul style="list-style-type: none"> • 0—(default) Ignore column correlations when creating the design. • 1—Consider the existing correlations in the tables when creating the design. • 2—Analyze column correlations if not previously performed, and consider the column correlations when creating the design. • 3—Analyze all column correlations in the tables and consider them when creating the design, even if they have been analyzed previously.

DISK_RESOURCE_REJECTIONS

Returns requests for resources that are rejected due to disk space shortages. Output is aggregated by both RESOURCE_TYPE and REJECTED_REASON to provide more comprehensive information.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
RESOURCE_TYPE	VARCHAR	The resource request requester (example: Temp files).
REJECTED_REASON	VARCHAR	One of the following: <ul style="list-style-type: none">• Insufficient disk space• Failed volume
REJECTED_COUNT	INTEGER	Number of times this REJECTED_REASON has been given for this RESOURCE_TYPE.
FIRST_REJECTED_TIMESTAMP	TIMESTAMP	The time of the first rejection for this REJECTED_REASON and RESOURCE_TYPE.
LAST_REJECTED_TIMESTAMP	TIMESTAMP	The time of the most recent rejection for this REJECTED_REASON and RESOURCE_TYPE.
LAST_REJECTED_VALUE	INTEGER	The value of the most recent rejection for this REJECTED_REASON and RESOURCE_TYPE.

See Also

- [RESOURCE_REJECTIONS](#)
- [CLEAR_RESOURCE_REJECTIONS](#)

DISK_STORAGE

Returns the amount of disk storage used by the database on each node. Each node can have one or more storage locations, and the locations can be on different disks with separate properties, such as free space, used space, and block size. The information in this system table is useful in determining where data files reside.

All returned values for this system table are in the context of the file system of the host operating system, and are not specific to Vertica-specific space.

The storage usage annotation called CATALOG indicates that the location is used to store the catalog. Each CATALOG location is specified only when creating a new database. You cannot add a CATALOG location annotation using [CREATE LOCATION](#), nor remove an existing CATALOG annotation.

Storage Location Performance

The performance of a storage location is measured with two values:

- Throughput in MB/sec
- Latency in seeks/sec

These two values are converted to a single number (Speed) with the following formula:

$\text{ReadTime (time to read 1MB)} = 1 / \text{throughput} + 1 / \text{latency}$

- $1/\text{throughput}$ is the time taken to read 1MB of data
- $1/\text{latency}$ is the time taken to seek to the data.
- ReadTime is the time taken to read 1MB of data.

A disk is faster than another disk if its ReadTime is less.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
STORAGE_PATH	VARCHAR	The path where the storage location is mounted.
STORAGE_USAGE	VARCHAR	The type of information stored in the location:

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • DATA: Only data is stored in the location. • TEMP: Only temporary files that are created during loads or queries are stored in the location. • DATA,TEMP: Both types of files are stored in the location. • USER: The storage location can be used by non-dbadmin users, who are granted access to the storage location • CATALOG: The area is used for the Vertica catalog. This usage is set internally and cannot be removed or changed.
RANK	INTEGER	The rank assigned to the storage location based on its performance. Ranks are used to create a storage locations on which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns. See Managing Storage Locations in the Administrator's Guide.
THROUGHPUT	INTEGER	The measure of a storage location's performance in MB/sec. 1/throughput is the time taken to read 1MB of data.
LATENCY	INTEGER	The measure of a storage location's performance in seeks/sec. 1/latency is the time taken to seek to the data.
STORAGE_STATUS	VARCHAR	The status of the storage location: active or retired.
DISK_BLOCK_SIZE_BYTES	INTEGER	The block size of the disk in bytes.
DISK_SPACE_USED_BLOCKS	INTEGER	The number of disk blocks in use.
DISK_SPACE_USED_MB	INTEGER	The number of megabytes of disk storage in use.
DISK_SPACE_FREE_BLOCKS	INTEGER	The number of free disk blocks available.
DISK_SPACE_FREE_MB	INTEGER	The number of megabytes of free storage available.

Column Name	Data Type	Description
DISK_SPACE_FREE_PERCENT	VARCHAR	The percentage of free disk space remaining.

ERROR_MESSAGES

Lists system error messages and warnings Vertica encounters while processing queries. Some errors occur when no transaction is in progress, so the transaction identifier or statement identifier columns might return NULL.

Column Name	Data Type	Description
EVENT_TIMESTAMP	TIMESTAMPTZ	Time when the row recorded the event.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_ID	INTEGER	Identifier of the user who received the error message.
USER_NAME	VARCHAR	Name of the user who received the error message at the time Vertica recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
REQUEST_ID	INTEGER	Unique identifier of the query request in the user session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.
ERROR_LEVEL	VARCHAR	Severity of the error, can be one of: <ul style="list-style-type: none"> LOG INFO

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • NOTICE • WARNING • ERROR • ROLLBACK • INTERNAL • FATAL • PANIC
ERROR_CODE	INTEGER	Error code that Vertica reports.
MESSAGE	VARCHAR	Textual output of the error message.
DETAIL	VARCHAR	Additional information about the error message, in greater detail.
HINT	VARCHAR	Actionable hint about the error. For example: HINT: Set the locale in this session to en_US@collation=binary using the command <code>"\locale en_US@collation=binary"</code>

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

EVENT_CONFIGURATIONS

Monitors the configuration of events.

Column Name	Data Type	Description
EVENT_ID	VARCHAR	The name of the event.
EVENT_DELIVERY_CHANNELS	VARCHAR	The delivery channel on which the event occurred.

EXECUTION_ENGINE_PROFILES

Provides profiling information about query execution runs. The hierarchy of IDs, from highest level to actual execution is:

- PATH_ID
- BASEPLAN_ID
- LOCALPLAN_ID
- OPERATOR_ID

Counters (output from the COUNTER_NAME column) are collected for each actual Execution Engine (EE) operator instance.

The following columns combine to form a unique key for rows in EXECUTION_ENGINE_PROFILES:

- TRANSACTION_ID
- STATEMENT_ID
- NODE_NAME
- OPERATOR_ID
- COUNTER_NAME
- COUNTER_TAG

For more about profiling and debugging, see [Profiling Database Performance](#) in the Administrator's Guide.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Node name for which information is listed.
USER_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the user.
USER_NAME	VARCHAR	User name for which query profile information is listed.
SESSION_ID	VARCHAR	Identifier of the session for which profiling information is

Column Name	Data Type	Description
		captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed.
OPERATOR_NAME	VARCHAR	Name of the Execution Engine (EE) component; for example, NetworkSend.
OPERATOR_ID	INTEGER	Identifier assigned by the EE operator instance that performs the work. OPERATOR_ID is different from LOCALPLAN_ID because each logical operator, such as Scan, may be executed by multiple threads concurrently. Each thread operates on a different operator instance, which has its own ID.
BASEPLAN_ID	INTEGER	Assigned by the optimizer on the initiator to EE operators in the original base (EXPLAIN) plan. Each EE operator in the base plan gets a unique ID.
PATH_ID	INTEGER	Identifier that Vertica assigns to a query operation or <i>path</i> ; for example to a logical grouping operation that might be performed by multiple execution engine operators. For each path, the same PATH ID is shared between the query plan (using EXPLAIN output) and in error messages that refer to joins.
LOCALPLAN_ID	INTEGER	Identifier assigned by each local executor while preparing for plan execution (local planning). Some operators in the base plan, such as the Root operator, which is connected to the client, do not run on all nodes. Similarly, certain operators, such as ExprEval, are added and removed during local planning due to implementation details.

Column Name	Data Type	Description
ACTIVITY_ID	INTEGER	Identifier of the plan activity.
RESOURCE_ID	INTEGER	Identifier of the plan resource.
COUNTER_NAME	VARCHAR	Name of the counter. See the "COUNTER_NAME Values" section below this table. The counter counts events for one statement.
COUNTER_TAG	VARCHAR	String that uniquely identifies the counter for operators that might need to distinguish between different instances. For example, COUNTER_TAG is used to identify to which of the node bytes are being sent to or received from for the NetworkSend operator.
COUNTER_VALUE	INTEGER	Value of the counter.
IS_EXECUTING	BOOLEAN	Indicates whether the profile is active or completed, where <i>t</i> is active and <i>f</i> is completed.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

COUNTER_NAME Values

The value of COUNTER_NAME can be any of the following:

COUNTER_NAME	Description
active threads	A counter of the LoadUnion operator, which indicates the number of input threads (Load operators) that are currently processing input.
blocks analyzed by SIPS expression	The number of data blocks analyzed by SIPS expression from the Scan operator.
blocks filtered by SIPS expression	The number of data blocks filtered by SIPS expression from the Scan operator.

COUNTER_NAME	Description
blocks filtered by SIPs value lists	The number of data blocks filtered by SIPs sorted value lists from the Scan operator.
buffers spilled	[NetworkSend] Buffers spilled to disk by NetworkSend.
bytes read from cache	[DataSource] The number of bytes read from Vertica cache when an EE DataSource operator is reading from ROS containers.
bytes read from disk	[DataSource] The number of bytes read from disk when an EE DataSource operator is reading from ROS containers.
bytes received	[NetworkRecv] The number of bytes received over the network for query execution.
bytes sent	[NetworkSend] Size of data after encoding and compression sent over the network (actual network bytes).
bytes spilled	[NetworkSend] Bytes spilled to disk by NetworkSend.
bytes total	Only relevant to SendFiles operator (that is, recover-by-container plan) total number of bytes to send / receive.
clock time (us)	Real-time clock time spent processing the query, in microseconds.
completed merge phases	Number of merge phases already completed by an LSort or DataTarget operator. Compare to the total merge phases. Variants on this value include join inner completed merge phases.
cumulative size of raw temp data (bytes)	Total amount of temporary data the operator has written to files. Compare to cumulative size of temp files (bytes) to understand impact of encoding and compression in an externalizing operator. Variants on this value include join inner cumulative size of raw temp files (bytes).

COUNTER_NAME	Description
cumulative size of temp files (bytes)	For externalizing operators only, the total number of encoded and compressed temp data the operator has written to files. A sort operator might go through multiple merge phases, where at each pass sorted chunks of data are merged into fewer chunks. This counter remembers the cumulative size of all temp files past and present. Variants on this value include <code>join inner cumulative size of temp files (bytes)</code> .
current allocated rid memory (bytes)	Per-rid memory tracking: current allocation amount under this rid.
current file handles	Number of files open.
current memory allocations (count)	Number of actual allocator calls made.
current memory capacity (bytes)	Amount of system memory held, which includes chunks which have been only partially consumed.
current memory overhead (bytes)	Memory consumed, for example, by debug headers. (Normally no overhead.)
current memory padding (bytes)	Memory padding for free list tiers (2^n bytes).
current memory requested (bytes)	Memory actually requested by the caller.
current size of temp files (bytes)	For externalizing operators only, the current size of the encoded and compressed temp data that the operator has written to files. Variants on this value include <code>join inner current size of temp files (bytes)</code> .
current threads	Unused.
current unbalanced memory allocations (count)	Pooled version of "current memory XXX" counters.
current unbalanced memory capacity (bytes)	

COUNTER_NAME	Description
current unbalanced memory overhead (bytes)	
current unbalanced memory requested (bytes)	
distinct value estimation time (μ s)	[Analyze Statistics] Time spent estimating the number of distinct values from the sample after data has been read off disk and into the statistical sample.
encoded bytes received	[NetworkRecv] Size of received data after decompressed (but still encoded) received over the network.
encoded bytes sent	[NetworkSend] Size of data sent over the network after encoding.
end time	Time (timestamp) when Vertica stopped processing the operation
estimated rows produced	Number of rows that the optimizer estimated would be produced. See rows produced for the actual number of rows that are produced.
exceptions cumulative size of raw temp data (bytes)	Counters that store the total or current size of exception data.
exceptions rows cumulative size of temp files (bytes)	
exceptions rows current size of temp files (bytes)	
execution time (us)	CPU clock time spent processing the query, in microseconds.
fast aggregated rows	The number of rows being processed by fast aggregations in the hash groupby operator (no group/aggregation).

COUNTER_NAME	Description
file handles	The number of file handles in use for an operator. Deprecated. See peak file handles or current file handles.
files completed	Relevant only to SendFiles/RecvFiles operators (that is, recover-by-container plan) number of files sent / received.
files total	Relevant only to SendFiles/RecvFiles operators (that is, recover-by-container plan) total number of files to send / receive.
Hadoop FS bytes read through native libhdfs++ client	[Scan, Load] The number of bytes read from an hdfs source (using libhdfs++).
Hadoop FS bytes read through webhdfs	[Scan, Load] The number of bytes read from a webhdfs source.
Hadoop FS bytes written through webhdfs	[DataTarget] The number of bytes written to webhdfs storage.
Hadoop FS hdfs:// operations that used native libhdfs++ calls	[Scan, Load, DataTarget] The number of times Vertica opened a file with an hdfs:// URL and used the native hdfs protocol
Hadoop FS hdfs:// operations that used webhdfs calls	[Scan, Load, DataTarget] The number of times Vertica opened a file with an hdfs:// URL and used the webhdfs protocol
Hadoop FS read operations through native libhdfs++ client failure count	[Scan, Load] The number of times a native libhdfs++ source encountered an error and gave up
Hadoop FS read operations through native libhdfs++ client retry count	[Scan, Load] The number of times a native libhdfs++ source encountered an error and retried
Hadoop FS read operations through webhdfs failure count	[Scan, Load] The number of times a webhdfs source encountered an error and gave up
Hadoop FS read operations	[Scan, Load] The number of times a webhdfs source

COUNTER_NAME	Description
through webhdfs retry count	encountered an error and retried
Hadoop FS write operations through webhdfs failure count	[DataTarget] The number of times a webhdfs write encountered an error and gave up
Hadoop FS write operations through webhdfs retry count	[DataTarget] The number of times a webhdfs write encountered an error and retried
histogram creation time (us)	[Analyze Statistics] Time spent estimating the number of distinct values from the sample after data has been read off disk and into the statistical sample.
initialization time (us)	The time in microseconds spent initializing an operator during the CompilePlan step of query processing. For example, initialization time could include the time spent compiling expressions and gathering resources.
input queue wait (μs)	Time in microseconds that an operator spends waiting for upstream operators.
input rows	Actual number of rows that were read into the operator.
input size (bytes)	Total number of bytes of the Load operator's input source, where NULL is unknown (read from FIFO).
inputs processed	The number of sources processed by a Load operator.
intermediate rows to process	The number of rows to be processed in a phase as determined by a sort or GROUP BY (HASH).
join inner clock time (us)	The real clock time spending on processing the inner input of the join operator.
join inner completed mergephases	See the completed merge phases counter.
join inner cumulative size of raw temp data (bytes)	

COUNTER_NAME	Description
join inner cumulative size of temp files (bytes)	
join inner current size of temp files (bytes)	
join inner execution time (us)	The CPU clock time spent on processing the inner input of the join operator.
join inner hash table building time (us)	The time spent for building the hash table for the inner input of the join operator.
join inner hash table collisions	The number of hash table collisions that occurred when building the hash table for the inner input of the join operator.
join inner hash table entries	The number of hash table entries for the inner input of the join operator.
join inner total merge phases	See the completed merge phases counter.
join outer clock time (us)	The real clock time spent on processing the outer input of the join operator (including doing the join).
join outer execution time (us)	The CPU clock time spent on processing the outer input of the join operator (including doing the join).
max sample size (rows)	[Analyze Statistics] Maximum number of rows that will be stored in the statistical sample.
memory allocated (bytes)	Memory allocated by this operator. Deprecated.
memory reserved (bytes)	Memory reserved by this operator. Deprecated.
network wait (us)	[NetworkSend, NetworkRecv] Time in microseconds spent waiting on the network.
number of cancel requests received	The number of cancel requests received (per operator) when cancelling a call to the execution engine.
number of invocations	The number of times a UDSF function was invoked.

COUNTER_NAME	Description
output queue wait (us)	Time in microseconds that an operator spends waiting for the output buffer to be consumed by a downstream operator.
peak allocated rid memory (bytes)	Per-rid memory tracking: peak allocation amount under this rid.
peak cooperating threads	Peak number of threads which parsed (in parallel) a single load source, using "cooperative parse." counter_tag indicates the source when joining with dc_load_events.
peak file handles	Peak value of the corresponding "current XXX" counters.
peak memory allocations (count)	
peak memory capacity (bytes)	
peak memory overhead (bytes)	
peak memory padding (bytes)	
peak memory requested (bytes)	
peak temp space	
peak threads	
peak unbalanced memory allocations (count)	
peak unbalanced memory capacity (bytes)	
peak unbalanced memory overhead (bytes)	
peak unbalanced memory padding (bytes)	

COUNTER_NAME	Description
peak unbalanced memory requested (bytes)	
portion offset	Offset value of a portion descriptor in an apportioned load. counter_tag indicates the source when joining with dc_load_events.
portion size	Size value of a portion descriptor in an apportioned load. counter_tag indicates the source when joining with dc_load_events.
producer stall (us)	[NetworkSend] Time in microseconds spent by NetworkSend when stalled waiting for network buffers to clear.
producer wait (us)	[NetworkSend] Time in microseconds spent by the input operator making rows to send.
read (bytes)	Number of bytes read from the input source by the Load operator.
receive time (us)	Time in microseconds that a Recv operator spends reading data from its socket.
rejected data cumulative size of raw temp data (bytes)	Counters that store total or current size of rejected row numbers. Are variants of: <ul style="list-style-type: none"> • cumulative size of raw temp data (bytes) • cumulative size of temp files (bytes) • current size of temp files (bytes)
rejected data cumulative size of temp files (bytes)	
rejected data current size of temp files (bytes)	
rejected rows cumulative size of raw temp data (bytes)	
rejected rows cumulative size of temp files (bytes)	

COUNTER_NAME	Description
rejected rows current size of temp files (bytes)	
reserved rid memory (bytes)	Per-rid memory tracking: total memory reservation under this rid.
rle rows produced	Number of physical tuples produced by an operator. Complements the rows produced counter, which shows the number of logical rows produced by an operator. For example, if a value occurs 1000 rows consecutively and is RLE encoded, it counts as 1000 rows produced not only 1 rle rows produced.
ROS blocks bounded	[DataTarget] Number of ROS blocks created, due to boundary alignment with RLE prefix columns, when an EE DataTarget operator is writing to ROS containers.
ROS blocks encoded	[DataTarget] Number of ros blocks created when an EE DataTarget operator is writing to ROS containers.
ROS bytes written	[DataTarget] Number of bytes written to disk when an EE DataTarget operator is writing to ROS containers.
rows filtered by SIPs expression	The number of rows filtered by the SIPs expression from the Scan operator.
rows in sample	[Analyze Statistics] Actual number of rows that will be stored in the statistical sample.
rows output by sort	[DataTarget] Number of rows sorted when an EE DataTarget operator is writing to ROS containers.
rows processed	[DataSource] Number of rows processed when an EE DataSource operator is reading from ROS containers.
rows processed by SIPs expression	The number of rows processed by the SIPs expression in the Scan operator.
rows produced	Number of logical rows produced by an operator. See also the rle rows produced counter.
rows pruned by valindex	[DataSource] Number of rows it skips direct scanning

COUNTER_NAME	Description
	with help of valindex when an EE DataSource operator is writing to ROS containers. This counter's value is not greater than "rows processed" counter.
rows read in sort	See the counter, total rows read in sort.
rows received	[NetworkRecv] Number of received sent over the network.
rows rejected	The number of rows rejected by the Load operator.
rows sent	[NetworkSend] Number of rows sent over the network.
rows to process	The total number of rows to be processed in a phase, based upon the number of table accesses. Compare to the counter, rows processed. Divide the rows processed value by the rows to process value for percent completion.
rows written in join sort	The total number of rows being read out of the sort facility in Join.
rows written in sort	The number of rows read out of the sort by the SortManager. This counter and the counter total rows read from sort are typically equal.
send time (us)	Time in microseconds that a Send operator spends writing data to its socket.
start time	Time (timestamp) when Vertica started to process the operation.
total merge phases	Number of merge phases an LSort or DataTarget operator must complete to finish sorting its data. NULL until the operator can compute this value (all data must first be ingested by the operator). Variants on this value include join inner total merge phases.
total rows read in join sort	The total number of rows being put into the sort facility in Join.
total rows read in sort	The total number of rows ingested into the sort by the

COUNTER_NAME	Description
total	SortManager. This counter and the counter rows written in sort are typically equal.
total rows written in sort	See the counter, rows written in sort.
total sources	Total number of distinct input sources processed in a load.
unpacked (bytes)	The number of bytes produced by a compressed source in a load (for example, for a gzip file, the size of the file when decompressed).
wait clock time (us)	StorageUnion wait time in microseconds.
WOS bytes acquired	Number of bytes acquired from the WOS by a DataTarget operator. Note: This is usually more but can be less than WOS bytes written if an earlier statement in the transaction acquired some WOS memory.
WOS bytes written	Number of bytes written to the WOS by a DataTarget operator.
written rows	[DataTarget] Number of rows written when an EE DataTarget operator writes to ROS containers

Examples

The two queries below show the contents of the EXECUTION_ENGINE_PROFILES table:

```
=> SELECT operator_name, operator_id, counter_name, counter_value
   FROM EXECUTION_ENGINE_PROFILES WHERE operator_name = 'Scan'
   ORDER BY counter_value DESC;
```

operator_name	operator_id	counter_name	counter_value
Scan	12	end time	397916465478595
Scan	9	end time	397916465478510
Scan	12	start time	397916465462098
Scan	9	start time	397916465447998
Scan	14	bytes read from disk	28044535
Scan	14	bytes read from disk	28030212
Scan	12	rows processed	5000000

```

Scan          |          12 | estimated rows produced |          4999999
Scan          |          18 | rows produced           |         1074828
Scan          |          18 | rle rows produced       |         1074828
Scan          |           3 | memory allocated (bytes) |        1074568
Scan          |           7 | rows produced           |         799526
Scan          |           7 | rle rows produced       |         799526
Scan          |           7 | memory allocated (bytes) |         682592
Scan          |          12 | clock time (us)         |        673806
Scan          |           7 | execution time (us)     |        545717
Scan          |           3 | memory allocated (bytes) |        537400
Scan          |          12 | clock time (us)         |        505315
Scan          |          14 | execution time (us)     |        495176
Scan          |           3 | bytes read from disk    |        452403
Scan          |          14 | execution time (us)     |        420189
Scan          |          12 | execution time (us)     |        404184
Scan          |          18 | clock time (us)         |        398751
Scan          |          18 | execution time (us)     |        339321
(24 rows)

```

```
=> SELECT DISTINCT counter_name FROM execution_engine_profiles;
           counter_name
```

```

-----
end time
clock time (us)
rle rows produced
bytes read from disk
start time
rows processed
memory allocated (bytes)
estimated rows produced
rows produced
execution time (us)
(10 rows)

```

The following query includes the `path_id` column, which links the path that the query optimizer takes (via the `EXPLAIN` command's textual output) with join error messages.

```
=> SELECT operator_name, path_id, counter_name, counter_value FROM execution_engine_profiles;
```

operator_name	path_id	counter_name	counter_value
Join	1	estimated rows produced	10000
Join	1	file handles	0
Join	1	memory allocated (bytes)	2405824
Join	1	memory reserved (bytes)	1769472
Join	1	rle rows produced	3
Join	1	rows produced	3
Join	1	clock time (us)	24105
Join	1	execution time (us)	235

EXTERNAL_TABLE_DETAILS

Returns the amount of disk storage used by the source files backing external tables in the database. The information in this system table is useful in determining Hadoop license

compliance.

When computing the size of an external table, Vertica counts all data found in the location specified by the COPY FROM clause. If you have a directory that contains ORC and delimited files, for example, and you define your external table with "COPY FROM *" instead of "COPY FROM *.orc", this table includes the size of the delimited files. (You would probably also encounter errors when querying that external table.) When you query this system table Vertica does not validate your table definition; it just uses the path to find files to report.

Restrict your queries to filter by schema, table, or format to avoid expensive queries. Vertica calculates the values in this table at query time, so "SELECT *" accesses every input file contributing to every external table.

Predicates in queries may use only the TABLE_SCHEMA, TABLE_NAME, and SOURCE_FORMAT columns. Values are case-sensitive.

This table includes TEMP external tables.

Column Name	Data Type	Description
SCHEMA_OID	INTEGER	The unique identification number of the schema in which the external table resides.
TABLE_SCHEMA	VARCHAR	The name of the schema in which the external table resides.
TABLE_OID	VARCHAR	A unique numeric ID assigned by the Vertica catalog that identifies the table.
TABLE_NAME	INTEGER	The table name.
SOURCE_FORMAT	VARCHAR	The data format the source file used, one of ORC, PARQUET, DELIMITED, USER DEFINED, or NULL if another format.
TOTAL_FILE_COUNT	INTEGER	The number of files used to store this table's data, expanding globs and partitions.
TOTAL_FILE_SIZE_BYTES	INTEGER	Total number of bytes used by all of this table's data files.
SOURCE_STATEMENT	VARCHAR	The load statement used to copy data from the source files.
FILE_ACCESS_ERROR	VARCHAR	The access error returned during the source statement. NULL, if there was no access error during the source

Column Name	Data Type	Description
		statement.

HOST_RESOURCES

Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.

Column Name	Data Type	Description
HOST_NAME	VARCHAR	The host name for which information is listed.
OPEN_FILES_LIMIT	INTEGER	The maximum number of files that can be open at one time on the node.
THREADS_LIMIT	INTEGER	The maximum number of threads that can coexist on the node.
CORE_FILE_LIMIT_MAX_SIZE_BYTES	INTEGER	The maximum core file size allowed on the node.
PROCESSOR_COUNT	INTEGER	The number of system processors.
PROCESSOR_CORE_COUNT	INTEGER	The number of processor cores in the system.
PROCESSOR_DESCRIPTION	VARCHAR	A description of the processor. For example: Inter(R) Core(TM)2 Duo CPU T8100 @2.10GHz (1 row)
OPENED_FILE_COUNT	INTEGER	The total number of open files on the node.
OPENED_SOCKET_COUNT	INTEGER	The total number of open sockets on the node.
OPENED_NONFILE_NONSOCKET_COUNT	INTEGER	The total number of <i>other</i> file descriptions open in which 'other' could be a directory or FIFO. It is not an open file or socket.
TOTAL_MEMORY_BYTES	INTEGER	The total amount of physical RAM, in bytes, available on the system.
TOTAL_MEMORY_FREE_BYTES	INTEGER	The amount of physical RAM, in bytes, left unused by the system.

Column Name	Data Type	Description
TOTAL_BUFFER_MEMORY_BYTES	INTEGER	The amount of physical RAM, in bytes, used for file buffers on the system
TOTAL_MEMORY_CACHE_BYTES	INTEGER	The amount of physical RAM, in bytes, used as cache memory on the system.
TOTAL_SWAP_MEMORY_BYTES	INTEGER	The total amount of swap memory available, in bytes, on the system.
TOTAL_SWAP_MEMORY_FREE_BYTES	INTEGER	The total amount of swap memory free, in bytes, on the system.
DISK_SPACE_FREE_MB	INTEGER	The free disk space available, in megabytes, for all storage location file systems (data directories).
DISK_SPACE_USED_MB	INTEGER	The disk space used, in megabytes, for all storage location file systems.
DISK_SPACE_TOTAL_MB	INTEGER	The total free disk space available, in megabytes, for all storage location file systems.

IO_USAGE

Provides disk I/O bandwidth usage history for the system.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
START_TIME	TIMESTAMP	Beginning of history interval.
END_TIME	TIMESTAMP	End of history interval.
READ_KBYTES_PER_SEC	FLOAT	Counter history of the number of bytes read measured in kilobytes per second.
WRITTEN_KBYTES_PER_SEC	FLOAT	Counter history of the number of bytes written measured in kilobytes per second.

Privileges

Superuser

LDAP_LINK_EVENTS

Monitors events that occurred during an LDAP Link synchronization.

Column Name	Data Type	Description
EVENT_TIMESTAMP	TIMESTAMP	The time the event occurred.
NODE_NAME	VARCHAR	The name of the node or nodes for which the information is listed.
SESSION_ID	VARCHAR	The identification number of the LDAP Link session.
USER_ID	INTEGER	The unique, system-generated user identification number.
USER_NAME	VARCHAR	The name of the user for which the information is listed.
TRANSACTION_ID	INTEGER	The system-generated transaction identification number. Is NULL if a transaction id does not exist.
EVENT_TYPE	VARCHAR	The type of event being logged, for example USER_CREATED and PROCESSING_STARTED.
ENTRY_NAME	VARCHAR	The name of the object on which the event occurred, if applicable. For example, the event SYNC-STARTED does not use an object.
ENTRY_OID	INTEGER	The unique identification number for the object on which the event occurred, if applicable.
LDAPURIHASH	INTEGER	The URI hash number for the LDAP user.

LOAD_SOURCES

Like [LOAD_STREAMS](#), monitors active and historical load metrics on each node. The `LOAD_SOURCES` table breaks information down by source and portion. Rows appear in this table only for `COPY` operations that are profiled or run for more than one second. `LOAD_SOURCES` does not record information about loads from ORC or Parquet files or `COPY LOCAL`.

A row is added to this table when the loading of a source or portion begins. Column values related to the progress of the load are updated during the load operation.

The columns uniquely identifying the load source (the various ID and name columns) and `IS_EXECUTING` always have non-NULL values.

Column Name	Data Type	Description
<code>SESSION_ID</code>	VARCHAR	Identifier of the session for which Vertica captures load stream information. This identifier is unique within the cluster for the current session but can be reused in a subsequent session.
<code>TRANSACTION_ID</code>	INTEGER	Identifier for the transaction within a session. If a session is active, but no transaction has begun, this value is NULL.
<code>STATEMENT_ID</code>	INTEGER	Unique numeric ID for the currently running statement. NULL indicates that no statement is currently being processed. The combination of <code>TRANSACTION_ID</code> , <code>STATEMENT_ID</code> uniquely identifies a statement within a session.
<code>STREAM_NAME</code>	VARCHAR	<p>Load stream identifier. If the user does not supply a specific name, the <code>STREAM_NAME</code> default value is <i>tablename-ID</i>, where:</p> <ul style="list-style-type: none"> <i>tablename</i> is the table into which data is being loaded. <i>ID</i> is an integer value. <i>ID</i> is guaranteed to be unique within the current session on a node. <p>This system table includes stream names for every <code>COPY</code> statement that takes more than 1 second to</p>

Column Name	Data Type	Description
		run. The 1-second duration includes the time to plan and execute the statement.
SCHEMA_NAME	VARCHAR	Schema name for which load information is listed. Lets you identify two streams that are targeted at tables with the same name in different schemas. NULL, if selecting from an external table.
TABLE_OID	INTEGER	A unique numeric ID assigned by the Vertica catalog that identifies the table. NULL, if selecting from an external table.
TABLE_NAME	VARCHAR	Name of the table being loaded. NULL, if selecting from an external table.
NODE_NAME	VARCHAR	Name of the node loading the source.
SOURCE_NAME	VARCHAR	<ul style="list-style-type: none"> • Full file path if copying from a file. • Value returned by <code>getUri()</code> if the source is a user-defined source. • STDIN if loading from standard input.
PORTION_OFFSET	INTEGER	Offset of the source portion, or NULL if not apportioned.
PORTION_SIZE	INTEGER	Size of the source portion, or NULL if not apportioned.
IS_EXECUTING	BOOLEAN	Whether this source is currently being parsed, where <i>t</i> is true and <i>f</i> is false.
READ_BYTES	INTEGER	Number of bytes read from the input file.
ROWS_PRODUCED	INTEGER	Number of rows produced from parsing the source.
ROWS_REJECTED	INTEGER	Number of rows rejected from parsing the source.
INPUT_SIZE	INTEGER	Size of the input source in bytes, or NULL for unsized sources. For UDSources, this value is the value returned by <code>getSize()</code> .

Column Name	Data Type	Description
PARSE_COMPLETE_PERCENT	INTEGER	Percent of rows from the input file that have been parsed.
FAILURE_REASON	VARCHAR	For load failures, error message indicating why loading from this source failed or was cancelled: <ul style="list-style-type: none"> • In cases of a node failure, exception, or cancellation, this column contains the associated error message. • If ERROR TOLERANCE was not used and a failure occurred, "Source aborted due to exception while loading from (name)". • For any other failure, "Exception occurred during load".
PEAK_COOPERATING_THREADS	INTEGER	The peak number of threads parsing this source in parallel.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Examples

The following example shows the values you would see in the LOAD_SOURCES table after some loads with various errors.

```
=> CREATE TABLE tt (i int);
=> COPY tt FROM '/tmp/load_sources_large.dat';
^Ccancel request sent
ERROR 3322: Execution canceled by operator

=> COPY tt FROM '/tmp/load_sources_large.dat' ON ANY NODE;
  Rows Loaded
  -----
  1000000
  (1 row)

=> COPY tt FROM '/tmp/load_sources_*.txt' ON ANY NODE;
```



```

initiator-9959:0x68 | 45035996273705528 |          4 |          | public |
45035996273743434 | tt          | e0          | /tmp/load_sources_large.dat | 1805556 |
5416672 | 1805556 | 2097152 | 1805558 | 164142 | 82070 |
100 | f |
initiator-9959:0x68 | 45035996273705528 |          4 |          | public |
45035996273743434 | tt          | e1          | /tmp/load_sources_large.dat | 1805558 |
0 | 1805558 | 2097152 | 1805559 | 179293 | 89646 |
| f |
initiator-9959:0x68 | 45035996273705528 |          4 |          | public |
45035996273743434 | tt          | e1          | /tmp/load_sources_large.dat | 1805557 |
1805558 | 1805557 | 2097152 | 1805563 | 164142 | 82071 |
100 | f |
initiator-9959:0x68 | 45035996273705528 |          4 |          | public |
45035996273743434 | tt          | initiator   | /tmp/load_sources_large.dat | 1805557 |
7222228 | 1805557 | 2097152 | 1805564 | 164142 | 82071 |
100 | f |
initiator-9959:0x68 | 45035996273705528 |          4 |          | public |
45035996273743434 | tt          | initiator   | /tmp/load_sources_large.dat | 1805556 |
9027785 | 1805556 | 1805556 | 1805556 | 164140 | 82071 |
100 | f |
initiator-9959:0x68 | 45035996273705532 |          1 |          | public |
45035996273743434 | tt          | e0          | /tmp/load_sources_2.txt | 7838 |
| f          | 7838 | 553 | 88 | 44 |
| Source aborted due to exception while loading from '/tmp/load_sources_perm.txt' on
node 'e1'
initiator-9959:0x68 | 45035996273705532 |          1 |          | public |
45035996273743434 | tt          | e1          | /tmp/load_sources_1.txt | 7838 |
| f          | 7838 | 215 | 36 | 18 |
| Source aborted due to exception while loading from '/tmp/load_sources_perm.txt' on
node 'e1'
initiator-9959:0x68 | 45035996273705532 |          1 |          | public |
45035996273743434 | tt          | e1          | /tmp/load_sources_perm.txt |
| f          | 0 | 0 | 0 | 0 |
| ERROR 2018: COPY: Could not open file [/tmp/load_sources_perm.txt] for reading;
Permission denied
initiator-9959:0x68 | 45035996273705532 |          1 |          | public |
45035996273743434 | tt          | initiator   | /tmp/load_sources_3.txt | 7838 |
| f          | 7838 | 2327 | 312 | 156 |
| Source aborted due to exception while loading from '/tmp/load_sources_perm.txt' on
node 'e1'
initiator-9959:0x68 | 45035996273705532 |          2 |          | public |
45035996273743434 | tt          | e0          | /tmp/load_sources_2.txt | 7838 |
| f          | 7838 | 7838 | 1000 | 500 |
initiator-9959:0x68 | 45035996273705532 |          2 |          | public |
45035996273743434 | tt          | e1          | /tmp/load_sources_1.txt | 7838 |
| f          | 7838 | 7838 | 1000 | 500 |
initiator-9959:0x68 | 45035996273705532 |          2 |          | public |
45035996273743434 | tt          | e1          | /tmp/load_sources_perm.txt |
| f          | 0 | 0 | 0 | 0 |
| ERROR 2018: COPY: Could not open file [/tmp/load_sources_perm.txt] for reading;
Permission denied
initiator-9959:0x68 | 45035996273705532 |          2 |          | public |
45035996273743434 | tt          | initiator   | /tmp/load_sources_3.txt | 7838 |
| f          | 7838 | 7838 | 1000 | 500 |
(22 rows)

```

LOAD_STREAMS

Monitors active and historical load metrics for load streams. This is useful for obtaining statistics about how many records got loaded and rejected from the previous load. Vertica maintains system table metrics until they reach a designated size quota (in kilobytes). This quota is set through internal processes, which you cannot set or view directly.

Column Name	Data Type	Description
SESSION_ID	VARCHAR	Identifier of the session for which Vertica captures load stream information. This identifier is unique within the cluster for the current session, but can be reused in a subsequent session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within a session. If a session is active but no transaction has begun, this is NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.
STREAM_NAME	VARCHAR	Load stream identifier. If the user does not supply a specific name, the STREAM_NAME default value is: <i>tablename-ID</i> where <i>tablename</i> is the table into which data is being loaded, and <i>ID</i> is an integer value, guaranteed to be unique with the current session on a node. This system table includes stream names for every COPY statement that takes more than 1-second to run. The 1-second duration includes the time to plan and execute the statement.
SCHEMA_NAME	VARCHAR	Schema name for which load stream information is listed. Lets you identify two streams that are targeted at tables with the same name in different schemas
TABLE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog

Column Name	Data Type	Description
		that identifies the table.
TABLE_NAME	VARCHAR	Name of the table being loaded.
LOAD_START	VARCHAR	Linux system time when the load started.
LOAD_DURATION_MS	NUMERIC (54,0)	Duration of the load stream in milliseconds.
IS_EXECUTING	BOOLEAN	Indicates whether the load is executing, where <i>t</i> is true and <i>f</i> is false.
ACCEPTED_ROW_COUNT	INTEGER	Number of rows loaded.
REJECTED_ROW_COUNT	INTEGER	Number of rows rejected.
READ_BYTES	INTEGER	Number of bytes read from the input file.
INPUT_FILE_SIZE_BYTES	INTEGER	Size of the input file in bytes. Note: When using STDIN as input, the input file size is zero (0).
PARSE_COMPLETE_PERCENT	INTEGER	Percent of rows from the input file that have been parsed.
UNSORTED_ROW_COUNT	INTEGER	Cumulative number rows not sorted across all projections. Note: UNSORTED_ROW_COUNT could be greater than ACCEPTED_ROW_COUNT because data is copied and sorted for every projection in the target table.
SORTED_ROW_COUNT	INTEGER	Cumulative number of rows sorted across all projections.
SORT_COMPLETE_PERCENT	INTEGER	Percent of rows from the input file that have been sorted.

Privileges

If you have the SYSMONITOR role or are the dbadmin user, this table shows all loads. Otherwise it shows only your loads.

LOCK_USAGE

Provides aggregate information about lock requests, releases, and attempts, such as wait time/count and hold time/count. Vertica records:

- Lock attempts at the end of the locking process
- Lock releases after lock attempts are released

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information on which lock interaction occurs.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
OBJECT_NAME	VARCHAR	Name of object being locked; can be a table or an internal structure (projection, global catalog, or local catalog).
MODE	VARCHAR	Intended operations of the transaction. Otherwise, this value is NONE. For a list of lock modes and compatibility, see Lock Modes .
AVG_HOLD_TIME	INTERVAL	Average time (measured in intervals) that Vertica holds a lock.
MAX_HOLD_TIME	INTERVAL	Maximum time (measured in intervals) that Vertica holds a lock.
HOLD_COUNT	INTEGER	Total number of times the lock was granted in the given mode.
AVG_WAIT_TIME	INTERVAL	Average time (measured in intervals) that Vertica waits on the lock.
MAX_WAIT_TIME	INTERVAL	Maximum time (measured in intervals) that Vertica waits on a lock.
WAIT_COUNT	INTEGER	Total number of times lock was unavailable at the time it was first requested.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

- [DUMP_LOCKTABLE](#)
- [LOCKS](#)
- [PROJECTION_REFRESHES](#)
- [SELECT](#)
- [SESSION_PROFILES](#)

LOCKS

Monitors lock grants and requests for all nodes. A table call with no results indicates that no locks are in use.

Column Name	Data Type	Description
NODE_NAMES	VARCHAR	Nodes on which lock interaction occurs. Node Rollup: NODE_NAMES are separated by commas. A transaction can have the same lock in the same mode in the same scope on multiple nodes. However, the transaction gets only one (1) line in the table.
OBJECT_NAME	VARCHAR	Name of object being locked; can be a table or an internal structure (projection, global catalog, or local catalog).
OBJECT_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog that identifies the object being locked.

Column Name	Data Type	Description
TRANSACTION_ID	VARCHAR	Identification of transaction within the session, if any; otherwise NULL. Useful for creating joins to other system tables.
TRANSACTION_DESCRIPTION	VARCHAR	Identification of transaction and associated description. Typically this query caused the transaction's creation.
LOCK_MODE	VARCHAR	Intended operation of the transaction. For a list of lock modes and compatibility, see Lock Modes
LOCK_SCOPE	VARCHAR	<p>Expected duration of the lock after it is granted. Before the lock is granted, Vertica lists the scope as REQUESTED.</p> <p>Once a lock has been granted, the following scopes are possible:</p> <ul style="list-style-type: none"> • STATEMENT_LOCALPLAN • STATEMENT_COMPILE • STATEMENT_EXECUTE • TRANSACTION_POSTCOMMIT • TRANSACTION <p>All scopes, other than TRANSACTION, are transient and are used only as part of normal query processing.</p>
REQUEST_TIMESTAMP	TIMESTAMP	Time when the transaction began waiting on the lock.
GRANT_TIMESTAMP	TIMESTAMP	<p>Time the transaction acquired or upgraded the lock:</p> <ul style="list-style-type: none"> • Return values are NULL until the grant occurs. • If the grant occurs immediately, values might be the same as REQUEST_TIMESTAMP.

See Also

- [Vertica Database Locks](#)
- [DUMP_LOCKTABLE](#)
- [LOCK_USAGE](#)
- [PROJECTION_REFRESHES](#)
- [SELECT](#)
- [SESSION_PROFILES](#)
- [TRANSACTIONS](#)

LOGIN_FAILURES

This system table lists failures for each failed login attempt. This information helps you determine if a user is having difficulty getting into the database or identify a possible intrusion attempt.

Column Name	Data Type	Description
LOGIN_TIMESTAMP	TIMESTAMPTZ	Time when Vertica recorded the login.
DATABASE_NAME	VARCHAR	The name of the database for the login attempt.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_NAME	VARCHAR	Name of the user whose login failed at the time Vertica recorded the session.
CLIENT_HOSTNAME	VARCHAR	Host name and port of the TCP socket from which the client connection was made. NULL if the session is internal.
CLIENT_PID	INTEGER	Identifier of the client process that issued this connection.

Column Name	Data Type	Description
		In some cases, the client process is on a different machine from the server.
CLIENT_VERSION	VARCHAR	Unused.
CLIENT_OS_USER_NAME	VARCHAR	The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.
AUTHENTICATION_METHOD	VARCHAR	<p>Name of the authentication method used to validate the client application or user who is trying to connect to the server using the database user name provided</p> <p>Valid values:</p> <ul style="list-style-type: none"> • Trust • Reject • GSS • LDAP • Ident • Hash • TLS <p>See Implementing Client Authentication in the Administrator's Guide for further information.</p>
CLIENT_AUTHENTICATION_NAME	VARCHAR	Locally created name of the client authentication method.
REASON	VARCHAR	<p>Description of login failure reason.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • INVALID USER • ACCOUNT LOCKED

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • REJECT • FAILED • INVALID AUTH METHOD • INVALID DATABASE

Privileges

Superuser

MEMORY_USAGE

Records system resource history for memory usage. This is useful for comparing memory that Vertica uses versus memory in use by the entire system.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
START_TIME	TIMESTAMP	Beginning of history interval.
END_TIME	TIMESTAMP	End of history interval.
AVERAGE_MEMORY_USAGE_PERCENT	FLOAT	Records the average memory usage in percent of total memory (0-100) during the history interval.

Privileges

Superuser

MONITORING_EVENTS

Reports significant events that can affect database performance and functionality if you do not address their root causes.

See [Monitoring Events](#) in the Administrator's Guide for details.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
EVENT_CODE	INTEGER	Numeric identifier that indicates the type of event. See Event Types in Monitoring Events in the Administrator's Guide for a list of event type codes.
EVENT_ID	INTEGER	Unique numeric ID that identifies the specific event.
EVENT_SEVERITY	VARCHAR	Severity of the event from highest to lowest. These events are based on standard syslog severity types: <ul style="list-style-type: none"> 0 – Emergency 1 – Alert 2 – Critical 3 – Error 4 – Warning 5 – Notice 6 – Info 7 – Debug
EVENT_POSTED_TIMESTAMP	TIMESTAMPTZ	When this event was posted.
EVENT_CLEARED_TIMESTAMP	TIMESTAMPTZ	When this event was cleared. Note: You can also query the ACTIVE_EVENTS system table to see events that have not been cleared.

Column Name	Data Type	Description
EVENT_EXPIRATION	TIMESTAMPTZ	Time at which this event expires. If the same event is posted again prior to its expiration time, this field gets updated to a new expiration time.
EVENT_CODE_DESCRIPTION	VARCHAR	Brief description of the event and details pertinent to the specific situation.
EVENT_PROBLEM_DESCRIPTION	VARCHAR	Generic description of the event.

Privileges

Superuser

See Also

[ACTIVE_EVENTS](#)

NETWORK_INTERFACES

Provides information about network interfaces on all Vertica nodes.

Column Name	Data Type	Description
NODE_ID	INTEGER	Unique identifier for the node that recorded the row.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
INTERFACE	VARCHAR	Network interface name.
IP_ADDRESS_FAMILY	VARCHAR	Network address protocol.
IP_ADDRESS	VARCHAR	IP address for this interface.
SUBNET	VARCHAR	IP subnet for this interface.
MASK	VARCHAR	IP network mask for this interface.
BROADCAST_ADDRESS	VARCHAR	IP broadcast address for this interface.

Privileges

None

NETWORK_USAGE

Provides network bandwidth usage history on the system. This is useful for determining if Vertica is using a large percentage of its available network bandwidth.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
START_TIME	TIMESTAMP	Beginning of history interval.
END_TIME	TIMESTAMP	End of history interval.
TX_KBYTES_PER_SEC	FLOAT	Counter history of outgoing (transmitting) usage in kilobytes per second.
RX_KBYTES_PER_SEC	FLOAT	Counter history of incoming (receiving) usage in kilobytes per second.

Privileges

Superuser

NODE_EVICTIONS

Monitors node evictions on the system.

Column Name	Data Type	Description
EVICTION_TIMESTAMP	TIMESTAMPTZ	Timestamp when the eviction request was made.
NODE_NAME	VARCHAR	The node name logging the information.

Column Name	Data Type	Description
EVICTED_NODE_NAME	VARCHAR	The node name of the evicted node.
EVICTED_NODE_ID	INT	The evicted node ID.
NODE_STATE_BEFORE_EVICTION	VARCHAR	The previous node state at the time of eviction.

NODE_RESOURCES

Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
HOST_NAME	VARCHAR	The hostname associated with a particular node.
PROCESS_SIZE_BYTES	INTEGER	The total size of the program.
PROCESS_RESIDENT_SET_SIZE_BYTES	INTEGER	The total number of bytes that the process has in memory.
PROCESS_SHARED_MEMORY_SIZE_BYTES	INTEGER	The amount of shared memory used.
PROCESS_TEXT_MEMORY_SIZE_BYTES	INTEGER	The total number of text bytes that the process has in physical memory. This does not include any shared libraries.
PROCESS_DATA_MEMORY_SIZE_BYTES	INTEGER	The amount of physical memory, in bytes, used for performing processes. This does not include the executable code.
PROCESS_LIBRARY_MEMORY_SIZE_BYTES	INTEGER	The total number of library bytes that the process has in physical memory.
PROCESS_DIRTY_MEMORY_SIZE_BYTES	INTEGER	The number of bytes that have been modified since they were last written to disk.

Column Name	Data Type	Description
SPREAD_HOST	VARCHAR	The node name of the spread host.
NODE_PORT	VARCHAR	The port used for intra-cluster communication.
DATA_PORT	VARCHAR	The port used by the Vertica client.

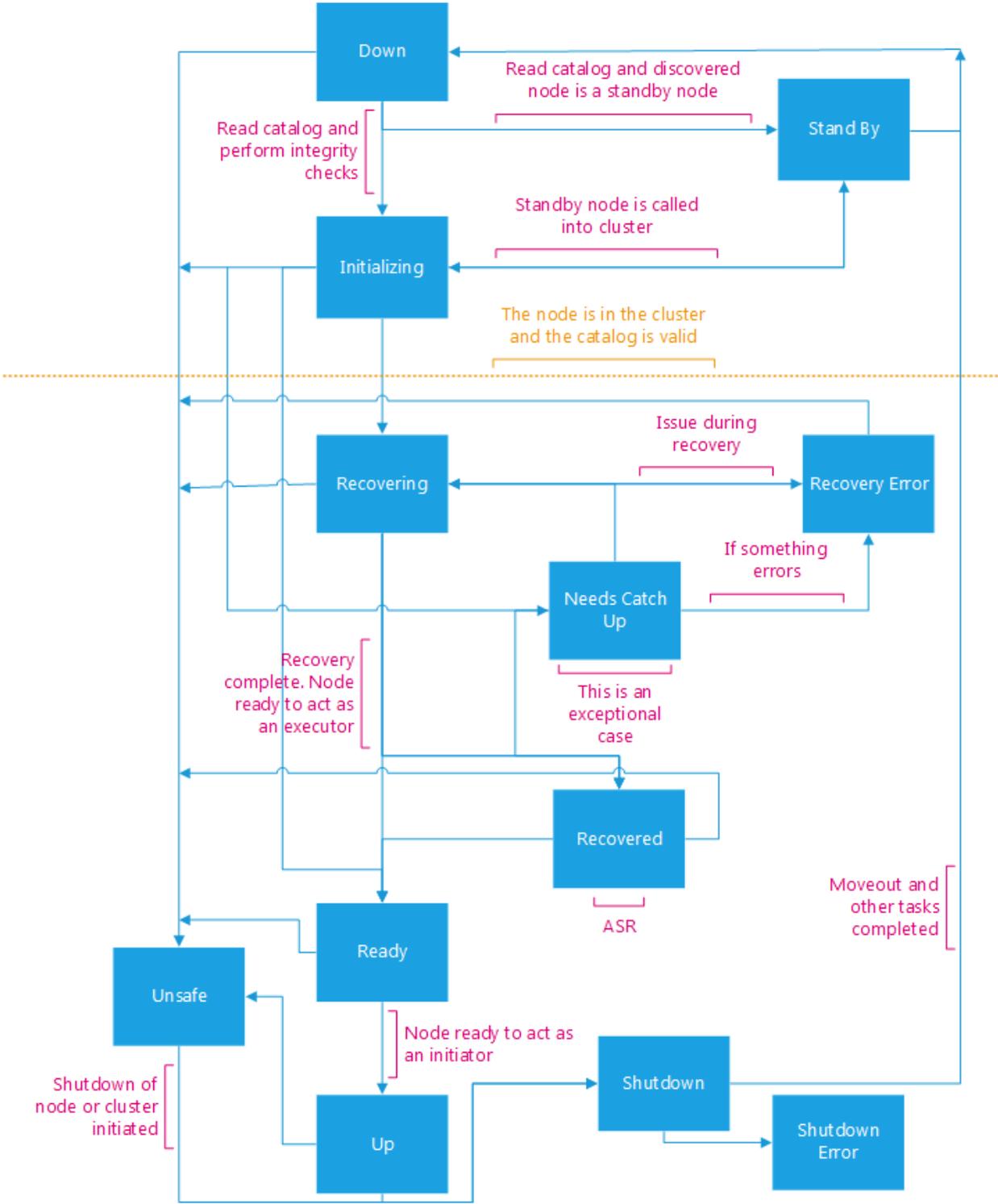
NODE_STATES

Monitors node recovery state-change history on the system.

Column Name	Data Type	Description
EVENT_TIMESTAMP	TIMESTAMPTZ	Time when Vertica recorded the event.
NODE_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the node.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
NODE_STATE	VARCHAR	Shows the node's state. Can be one of: <ul style="list-style-type: none"> • UP • DOWN • READY • UNSAFE • SHUTDOWN • SHUTDOWN ERROR • RECOVERING • RECOVERY ERROR • RECOVERED • INITIALIZING

Column Name	Data Type	Description
		<ul style="list-style-type: none"><li data-bbox="695 285 857 317">• STAND BY<li data-bbox="695 354 954 386">• NEEDS CATCH UP

The following flow chart details different node states:



Privileges

None

NOTIFIER_ERRORS

Reports errors encountered by [notifiers](#).

Column Name	Data Type	Description
ERROR_TIME	TIMESTAMPZ	The time that the error occurred.
NODE_NAME	VARCHAR	Name of the node that encountered the error.
NOTIFIER_NAME	VARCHAR	Name of the notifier that triggered the error.
DESCRIPTION	VARCHAR	A description of the error.

Privileges

Superuser

OUTPUT_DEPLOYMENT_STATUS

Contains information about the deployment status of all the projections in your design. Each row contains information about a different projection. Vertica populates this table when you deploy the database design by running the function [DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY](#).

Column Name	Column Type	Description
deployment_id	INTEGER	Unique ID that Database Designer assigned to the deployment.
design_name	VARCHAR	Unique name that the user assigned to the design.
deployment_projection_id	INTEGER	Unique ID that Database Designer assigned to the output projection.
deployment_projection_name	VARCHAR	Name that Database Designer assigned to the output projection or the name of the projection to be dropped.

Column Name	Column Type	Description
deployment_status	VARCHAR	Status of the deployment: <ul style="list-style-type: none"> • pending • complete • needs_refresh • in_progress • error
error_message	VARCHAR	Text of any error that occurred when creating or refreshing the specified projection.

OUTPUT_EVENT_HISTORY

Contains information about each stage that Database Designer performs to design and optimize your database design.

Column Name	Column Type	Description
TIME_STAMP	TIMESTAMP	Date and time of the specified stage.
DESIGN_ID	INTEGER	Unique id that Database Designer assigned to the design.
DESIGN_NAME	VARCHAR	Unique name that the user assigned to the design.
STAGE_TYPE	VARCHAR	Design stage that Database Designer was working on at the time indicated by the TIME_STAMP field. Possible values include: <ul style="list-style-type: none"> • Design in progress • Analyzing data statistics • Optimizing query performance • Optimizing storage footprint

Column Name	Column Type	Description
		<ul style="list-style-type: none"> • All done • Deployment in progress
ITERATION_NUMBER	INTEGER	Iteration number for the Optimizing query performance stage.
TOTAL_QUERY_COUNT	INTEGER	Total number of design queries in the design.
REMAINING_QUERY_COUNT	INTEGER	Number of design queries remaining for Database Designer to process.
MAX_STEP_NUMBER	INTEGER	Number of steps in the current stage.
CURRENT_STEP_NUMBER	INTEGER	Step in the current stage being processed at the time indicated by the TIME_STAMP field.
CURRENT_STEP_DESCRIPTION	VARCHAR	<p>Name of the step that Database Designer is performing at that time indicated in the TIME_STAMP field. Possible values include:</p> <ul style="list-style-type: none"> • Design with deployment started • Design in progress: Analyze statistics phase • design_table_name • projection_name • Design in progress: Query optimization phase • Extracting interesting columns • Enumerating sort orders • Setting up projection candidates • Assessing projection candidates • Choosing best projections

Column Name	Column Type	Description
		<ul style="list-style-type: none"> • Calculating estimated benefit of best projections • Complete • Design in progress: Storage optimization phase • Design completed successfully • Setting up deployment metadata • Identifying projections to be dropped • Running deployment • Deployment completed successfully
TABLE_ID	INTEGER	Unique id that Database Designer assigned to the design table.

Example

The following example shows the steps that Database Designer performs while optimizing the VMart example database:

```
=> SELECT DESIGNER_CREATE_DESIGN('VMART_DESIGN');
=> SELECT DESIGNER_ADD_DESIGN_TABLES('VMART_DESIGN','public.*');
=> SELECT DESIGNER_ADD_DESIGN_QUERIES('VMART_DESIGN','/tmp/examples/vmart_queries.sql',);
...
=> \x
Expanded display is on.
=> SELECT * FROM OUTPUT_EVENT_HISTORY;
-[ RECORD 1 ] -----
time_stamp          | 2013-06-05 11:44:41.588
design_id            | 45035996273705090
design_name          | VMART_DESIGN
stage_type          | Design in progress
iteration_number     |
total_query_count   |
remaining_query_count |
max_step_number     |
current_step_number |
current_step_description | Design with deployment started
table id            |
```

```

-[ RECORD 2 ] -----+-----
time_stamp          | 2013-06-05 11:44:41.611
design_id            | 45035996273705090
design_name          | VMART_DESIGN
stage_type          | Design in progress
iteration_number     |
total_query_count   |
remaining_query_count |
max_step_number     |
current_step_number |
current_step_description | Design in progress: Analyze statistics phase
table id            |
-[ RECORD 3 ] -----+-----
time_stamp          | 2013-06-05 11:44:42.011
design_id            | 45035996273705090
design_name          | VMART_DESIGN
stage_type          | Analyzing statistics
iteration_number     |
total_query_count   |
remaining_query_count |
max_step_number     | 15
current_step_number | 1
current_step_description | public.customer_dimension
table id            |
...
-[ RECORD 20 ] -----+-----
time_stamp          | 2013-06-05 11:44:49.324
design_id            | 45035996273705090
design_name          | VMART_DESIGN
stage_type          | Optimizing query performance
iteration_number     | 1
total_query_count   | 9
remaining_query_count | 9
max_step_number     | 7
current_step_number | 1
current_step_description | Extracting interesting columns
table id            |
...
-[ RECORD 62 ] -----+-----
time_stamp          | 2013-06-05 11:51:23.790
design_id            | 45035996273705090
design_name          | VMART_DESIGN
stage_type          | Deployment in progress
iteration_number     |
total_query_count   |
remaining_query_count |
max_step_number     |
current_step_number |
current_step_description | Deployment completed successfully
table id            |

```

PARTITION_COLUMNS

For each projection of each partitioned table, shows the disk space used by each column on each node. The column `disk_space_bytes` shows how much disk space the partitioned data uses, including deleted data. So, if you delete rows but do not purge them, the view's

`deleted_row_count` column changes to show the number of deleted rows in each column; however, `disk_space_bytes` remains the same. After the deleted rows are purged, Vertica reclaims the disk space: `disk_space_bytes` changes accordingly, and `deleted_row_count` is reset to 0.

For grouped partitions, `PARTITION_COLUMNS` shows the cumulative disk space used for each column per grouped partition. The column `grouped_partition_key`, if not null, identifies the partition in which a given column is grouped.

Column Name	Data Type	Description
<code>COLUMN_NAME</code>	VARCHAR	Identifies a named column within the partitioned table.
<code>COLUMN_ID</code>	INTEGER	Unique numeric ID assigned by the Vertica, which identifies the column.
<code>TABLE_NAME</code>	VARCHAR	Name of the partitioned table.
<code>PROJECTION_NAME</code>	VARCHAR	Projection name for which information is listed.
<code>PROJECTION_ID</code>	INTEGER	Unique numeric ID assigned by Vertica, which identifies the projection.
<code>NODE_NAME</code>	VARCHAR	The node that hosts partitioned data.
<code>PARTITION_KEY</code>	VARCHAR	Identifies the table partition.
<code>GROUPED_PARTITION_KEY</code>	VARCHAR	Identifies the grouped partition to which a given column belongs.
<code>ROW_COUNT</code>	INTEGER	The total number of partitioned data rows for each column, including deleted rows.
<code>DELETED_ROW_COUNT</code>	INTEGER	The number of deleted partitioned data rows in each column.
<code>DISK_SPACE_BYTES</code>	INTEGER	The amount of space used by partitioned data.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Example

Given the following table definition:

```
=> CREATE TABLE messages
(
    time_interval timestamp NOT NULL,
    thread_id varchar(32) NOT NULL,
    unique_id varchar(53) NOT NULL,
    msg_id varchar(65),
    ...
)
PARTITION BY ((messages.time_interval)::date);
```

a query on `partition_columns` might return the following (truncated) results:

```
=> SELECT * FROM partition_columns order by table_name, column_name;
```

column_name	column_id	table_name	projection_name	projection_id	node_name	partition_key	grouped_partition_key	row_count	deleted_row_count	disk_space_bytes
msg_id	45035996273743190	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-03		6147	0	41145
msg_id	45035996273743190	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-15		178	0	65
msg_id	45035996273743190	messages	messages_super	45035996273743182	v_vmart_node0003	2010-07-03		6782	0	45107
msg_id	45035996273743190	messages	messages_super	45035996273743182	v_vmart_node0003	2010-07-04		866	0	5883
...										
thread_id	45035996273743186	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-03		6147	0	70565
thread_id	45035996273743186	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-15		178	0	2429
thread_id	45035996273743186	messages	messages_super	45035996273743182	v_vmart_node0003	2010-07-03		6782	0	77730
thread_id	45035996273743186	messages	messages_super	45035996273743182	v_vmart_node0003	2010-07-04		866	0	10317
...										
time_interval	45035996273743184	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-03		6147	0	6320
time_interval	45035996273743184	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-15		178	0	265
time_interval	45035996273743184	messages	messages_super	45035996273743182	v_vmart_node0003	2010-07-03		6782	0	6967
time_interval	45035996273743184	messages	messages_super	45035996273743182	v_vmart_node0003	2010-07-04		866	0	892
...										
unique_id	45035996273743188	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-03		6147	0	70747
unique_id	45035996273743188	messages	messages_super	45035996273743182	v_vmart_node0002	2010-07-15		178	0	2460

```

unique_id | 45035996273743188 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
2010-07-03 | | 6782 | 0 | 77959 |
unique_id | 45035996273743188 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
2010-07-04 | | 866 | 0 | 10332 |
unique_id | 45035996273743188 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
2010-07-15 | | 184 | 0 | 2549 |
...
(11747 rows)

```

PARTITION_REORGANIZE_ERRORS

new column projection_id

Monitors all background partitioning tasks, and if Vertica encounters an error, creates an entry in this table with the appropriate information. Does not log repartitioning tasks that complete successfully.

Column Name	Data Type	Description
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
USER_NAME	VARCHAR	Name of the user who received the error at the time Vertica recorded the session.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
TABLE_NAME	VARCHAR	Name of the partitioned table.
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	Projection name for which information is listed.
MESSAGE	VARCHAR	Textual output of the error message.
HINT	VARCHAR	Actionable hint about the error.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

PARTITION_STATUS

For each projection of each partitioned table, shows the fraction of its data that is actually partitioned according to the current partition expression. When the partitioning of a table is altered, the value in `PARTITION_REORGANIZE_PERCENT` for each of its projections drops to zero and goes back up to 100 when all the data is repartitioned.

Column Name	Data Type	Description
<code>PROJECTION_ID</code>	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
<code>TABLE_SCHEMA</code>	VARCHAR	Name of the schema that contains the partitioned table.
<code>TABLE_NAME</code>	VARCHAR	Table name that is partitioned.
<code>TABLE_ID</code>	INTEGER	Unique numeric ID assigned by the Vertica, which identifies the table.
<code>PROJECTION_SCHEMA</code>	VARCHAR	Schema containing the projection.
<code>PROJECTION_NAME</code>	VARCHAR	Projection name for which information is listed.
<code>PARTITION_REORGANIZE_PERCENT</code>	INTEGER	For each projection, drops to zero and goes back up to 100 when all the data is repartitioned after the partitioning of a table has been altered. Ideally all rows will show 100 (%).

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

PARTITIONS

Displays partition metadata, one row per partition key, per ROS container.

Column Name	Data Type	Description
PARTITION_KEY	VARCHAR	The partition value(s).
PROJECTION_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
TABLE_SCHEMA	VARCHAR	The schema name for which information is listed.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
ROS_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the ROS container.
ROS_SIZE_BYTES	INTEGER	The ROS container size in bytes.
ROS_ROW_COUNT	INTEGER	Number of rows in the ROS container.
NODE_NAME	VARCHAR	Node where the ROS container resides.
DELETED_ROW_COUNT	INTEGER	The number of rows in the partition.
LOCATION_LABEL	VARCHAR	The location label of the default storage location.

Notes

- A many-to-many relationship exists between partitions and ROS containers. PARTITIONS displays information in a denormalized fashion.
- To find the number of ROS containers having data of a specific partition, aggregate PARTITIONS over the `partition_key` column.
- To find the number of partitions stored in a ROS container, aggregate PARTITIONS over the `ros_id` column.

Example

Given the unsegmented projection `states_p` replicated across three nodes, the following query on the `PARTITIONS` table returns twelve rows, representing twelve ROS containers:

```
=> SELECT PARTITION_KEY, ROS_ID, ROS_SIZE_BYTES, ROS_ROW_COUNT, NODE_NAME FROM partitions WHERE
PROJECTION_NAME='states_p' order by ROS_ID;
```

PARTITION_KEY	ROS_ID	ROS_SIZE_BYTES	ROS_ROW_COUNT	NODE_NAME
VT	45035996281231297	95	14	v_vmart_node0001
PA	45035996281231309	92	11	v_vmart_node0001
NY	45035996281231321	90	9	v_vmart_node0001
MA	45035996281231333	96	15	v_vmart_node0001
VT	49539595902704977	95	14	v_vmart_node0002
PA	49539595902704989	92	11	v_vmart_node0002
NY	49539595902705001	90	9	v_vmart_node0002
MA	49539595902705013	96	15	v_vmart_node0002
VT	54043195530075651	95	14	v_vmart_node0003
PA	54043195530075663	92	11	v_vmart_node0003
NY	54043195530075675	90	9	v_vmart_node0003
MA	54043195530075687	96	15	v_vmart_node0003

(12 rows)

PROCESS_SIGNALS

Returns a history of signals that were received and handled by the Vertica process. For details about signals, see the [Linux documentation](#).

Column Name	Data Type	Description
SIGNAL_TIMESTAMP	TIMESTAMPTZ	Time when Vertica recorded the signal.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
SIGNAL_NUMBER	INTEGER	Signal number, refers to POSIX SIGNAL_NUMBER
SIGNAL_CODE	INTEGER	Signal code.
SIGNAL_PID	INTEGER	Linux process identifier of the signal.
SIGNAL_UID	INTEGER	Process ID of sending process.
SIGNAL_ADDRESS	INTEGER	Address at which fault occurred.

Privileges

Superuser

PROJECTION_RECOVERIES

Retains history about projection recoveries. Because Vertica adds an entry per recovery plan, a projection/node pair might appear multiple times in the output.

Note: You cannot query this or other system tables during cluster recovery; the cluster must be UP to accept connections.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is recovering or has recovered the corresponding projection.
PROJECTION_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	Name of the projection that is being or has been recovered on the corresponding node.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any. TRANSACTION_ID initializes as NO_TRANSACTION with a value of 0. Vertica will ignore the recovery query and keep (0) if there's no action to take (no data in the table, etc). When no recovery transaction starts, ignored value appears in this table's STATUS column.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.
METHOD	VARCHAR	Recovery method that Vertica chooses. Possible values are: <ul style="list-style-type: none">incremental

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • incremental-replay-delete • split • recovery-by-container
STATUS	VARCHAR	<p>Current projection-recovery status on the corresponding node. STATUS can be "queued," which indicates a brief period between the time the query is prepared and when it runs. Possible values are:</p> <ul style="list-style-type: none"> • queued • running • finished • ignored • error-retry • error-fatal
PROGRESS	INTEGER	<p>An estimate (value in the range [0,100]) of percent complete for the recovery task described by this information.</p> <p>Note: The actual amount of time it takes to complete a recovery task depends on a number of factors, including concurrent workloads and characteristics of the data; therefore, accuracy of this estimate can vary.</p> <p>The PROGRESS column value is NULL after the task completes.</p>
DETAIL	VARCHAR	<p>More detailed information about PROGRESS. The values returned for this column depend on the type of recovery plan:</p> <ul style="list-style-type: none"> • General recovery plans – value displays the estimated progress, as a percent, of the three primary parts of the plan: Scan, Sort, and Write.

Column Name	Data Type	Description
		<ul style="list-style-type: none"> Recovery-by-container plans – value begins with CopyStorage: and is followed by the number of bytes copied over the total number of bytes to copy. Replay delete plans – value begins with Delete: and is followed by the number of deletes replayed over an estimate of the total number of deletes to replay. <p>The DETAIL column value becomes NULL after the recovery plan completes.</p>
START_TIME	TIMESTAMPTZ	Time the recovery task described by this information started.
END_TIME	TIMESTAMPTZ	Time the recovery task described by this information ended.
RUNTIME_PRIORITY	VARCHAR	<p>Determines the amount of runtime resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are:</p> <ul style="list-style-type: none"> HIGH MEDIUM LOW

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

[RECOVERY_STATUS](#)

PROJECTION_REFRESHES

Provides information about refresh operations for projections. This table retains information about a refresh operation—whether successful or unsuccessful—until the function [CLEAR_PROJECTION_REFRESHES](#) executes, or the storage quota for the table is exceeded.

Tables and projections can be dropped while a query runs against them. The query continues to run, even after the drop occurs. Only when the query finishes does it notice the drop, which could cause a rollback. The same is true for refresh queries. Thus, PROJECTION_REFRESHES might report that a projection failed to be refreshed before the refresh query completes. In this case, the REFRESH_DURATION_SEC column continues to increase until the refresh query completes.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Node where the refresh was initiated.
PROJECTION_SCHEMA	VARCHAR	Name of the schema associated with the projection.
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	Name of the projection that is targeted for refresh.
ANCHOR_TABLE_NAME	VARCHAR	Name of the projection's associated anchor table.
REFRESH_STATUS	VARCHAR	Status of the projection: <ul style="list-style-type: none"> • Queued — Indicates that a projection is queued for refresh. • Refreshing — Indicates that a refresh for a projection is in process. • Refreshed — Indicates that a refresh for a projection has successfully completed. • Failed — Indicates that a refresh for a projection did not successfully complete.
REFRESH_PHASE	VARCHAR	Indicates how far the refresh has progressed:

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • Historical – Indicates that the refresh has reached the first phase and is refreshing data from historical data. This refresh phase requires the most amount of time. • Current – Indicates that the refresh has reached the final phase and is attempting to refresh data from the current epoch. To complete this phase, refresh must be able to obtain a lock on the table. If the table is locked by some other transaction, refresh is put on hold until that transaction completes. <p>The LOCKS system table is useful for determining if a refresh has been blocked on a table lock. To determine if a refresh has been blocked, locate the term "refresh" in the transaction description. A refresh has been blocked when the scope for the refresh is REQUESTED and one or more other transactions have acquired a lock on the table.</p> <p>Note: The REFRESH_PHASE field is NULL until the projection starts to refresh and is NULL after the refresh completes.</p>
REFRESH_METHOD	VARCHAR	<p>Method used to refresh the projection:</p> <ul style="list-style-type: none"> • Buddy – Uses the contents of a buddy to refresh the projection. This method maintains historical data. This enables the projection to be used for historical queries. • Scratch – Refreshes the projection without using a buddy. This method does not generate historical data. This means that the projection cannot participate in historical queries from any point before the projection was refreshed. • Rebalance – If the projection is segmented it is refreshed from scratch; if unsegmented it is refreshed from buddy.

Column Name	Data Type	Description
REFRESH_FAILURE_COUNT	INTEGER	Number of times a refresh failed for the projection. FAILURE_COUNT does not indicate whether the projection was eventually refreshed. See REFRESH_STATUS to determine how the refresh operation is progressing.
SESSION_ID	VARCHAR	Unique numeric ID assigned by the Vertica catalog, which identifies the refresh session.
REFRESH_START	TIMESTAMPTZ	Time the projection refresh started (provided as a timestamp).
REFRESH_DURATION_SEC	INTERVAL SECOND (0)	Length of time that the projection refresh ran in seconds.
IS_EXECUTING	BOOLEAN	Distinguishes between active (<i>t</i>) and completed (<i>f</i>) refresh operations.
RUNTIME_PRIORITY	VARCHAR	Determines the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are: <ul style="list-style-type: none"> • HIGH • MEDIUM • LOW
transaction_id	Int	Identifier for the transaction within the session, if any; otherwise NULL. <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note: The <code>transaction_id</code> is correlated with the execution plan only when refreshing from scratch. When refreshing from a buddy, multiple sub-transactions are created</p> </div>

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

PROJECTION_STORAGE

Monitors the amount of disk storage used by each projection on each node.

Note: Projections that have no data never have full statistics. Querying this system table lets you see if your projection contains data.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
PROJECTION_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed.
PROJECTION_SCHEMA	VARCHAR	The name of the schema associated with the projection.
PROJECTION_COLUMN_COUNT	INTEGER	The number of columns in the projection.
ROW_COUNT	INTEGER	The number of rows in the table's projections, including any rows marked for deletion.
USED_BYTES	INTEGER	The number of bytes of disk storage used by the projection.
WOS_ROW_COUNT	INTEGER	The number of WOS rows in the projection.
WOS_USED_BYTES	INTEGER	The number of WOS bytes in the projection.
ROS_ROW_COUNT	INTEGER	The number of ROS rows in the projection.
ROS_USED_BYTES	INTEGER	The number of ROS bytes in the projection.
ROS_COUNT	INTEGER	The number of ROS containers in the projection.

Column Name	Data Type	Description
ANCHOR_TABLE_NAME	VARCHAR	The associated table name for which information is listed.
ANCHOR_TABLE_SCHEMA	VARCHAR	The associated table schema for which information is listed.
ANCHOR_TABLE_ID	INTEGER	A unique numeric ID, assigned by the Vertica catalog, which identifies the anchor table.

See Also

- [PROJECTIONS](#)
- [ANALYZE_STATISTICS](#)

PROJECTION_USAGE

Records information about projections Vertica used in each processed query.

Column Name	Data Type	Description
QUERY_START_TIMESTAMP	TIMESTAMPTZ	Value of query at beginning of history interval.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_NAME	VARCHAR	Name of the user at the time Vertica recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
REQUEST_ID	INTEGER	Unique identifier of the query request in the user session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running

Column Name	Data Type	Description
		statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.
IO_TYPE	VARCHAR	Input/output.
PROJECTION_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	Projection name for which information is listed.
ANCHOR_TABLE_ID	INTEGER	Unique numeric ID assigned by the Vertica, which identifies the anchor table.
ANCHOR_TABLE_SCHEMA	VARCHAR	Name of the schema that contains the anchor table.
ANCHOR_TABLE_NAME	VARCHAR	Name of the projection's associated anchor table.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

QUERY_EVENTS

Returns information about query planning, optimization, and execution events.

Column Name	Data Type	Description
EVENT_TIMESTAMP	TIMESTAMPTZ	Time when Vertica recorded the event.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_ID	INTEGER	Identifier of the user for the query event.
USER_NAME	VARCHAR	Name of the user for which Vertica lists query

Column Name	Data Type	Description
		information at the time it recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
REQUEST_ID	INTEGER	Unique identifier of the query request in the user session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.
EVENT_CATEGORY	VARCHAR	Category of event: OPTIMIZATION or EXECUTION.
EVENT_TYPE	VARCHAR	Type of event. For a list of events and their descriptions, see Check Query Events Proactively .
EVENT_DESCRIPTION	VARCHAR	Generic description of the event.
OPERATOR_NAME	VARCHAR	Name of the Execution Engine component that generated the event, if applicable; for example, NetworkSend. Values from the OPERATOR_NAME and PATH_ID columns let you tie a query event back to a particular operator in the query plan. If the event did not come from a specific operator, the OPERATOR_NAME column is NULL.
PATH_ID	INTEGER	Unique identifier that Vertica assigns to a query operation or path in a query plan. If the event did not come from a specific operator, the PATH_ID column is NULL. See EXECUTION_ENGINE_PROFILES for more information.
OBJECT_ID	INTEGER	Object identifier (such as projection or table) to which the event refers.

Column Name	Data Type	Description
EVENT_DETAILS	VARCHAR	Free-form text describing the specific event.
SUGGESTED_ACTION	VARCHAR	Suggested user action, if any is available.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

- [EXECUTION_ENGINE_PROFILES](#)
- [QUERY_PLAN_PROFILES](#)

QUERY_METRICS

Monitors the sessions and queries running on each node.

Note: Totals in this table are reset each time the database restarts.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
ACTIVE_USER_SESSION_COUNT	INTEGER	The number of active user sessions (connections).
ACTIVE_SYSTEM_SESSION_COUNT	INTEGER	The number of active system sessions.
TOTAL_USER_SESSION_COUNT	INTEGER	The total number of user sessions.
TOTAL_SYSTEM_SESSION_COUNT	INTEGER	The total number of system sessions.
TOTAL_ACTIVE_SESSION_COUNT	INTEGER	The total number of active user and system sessions.
TOTAL_SESSION_COUNT	INTEGER	The total number of user and system sessions.

Column Name	Data Type	Description
RUNNING_QUERY_COUNT	INTEGER	The number of queries currently running.
EXECUTED_QUERY_COUNT	INTEGER	The total number of queries that ran.

QUERY_PLAN_PROFILES

Provides detailed execution status for queries that are currently running in the system. Output from the table shows the real-time flow of data and the time and resources consumed for each path in each query plan.

Column Name	Data Type	Description
TRANSACTION_ID	INTEGER	An identifier for the transaction within the session if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session; these columns are useful for creating joins with other system tables.
PATH_ID	INTEGER	Unique identifier that Vertica assigns to a query operation or path in a query plan. Textual representation for this path is output in the PATH_LINE column.
PATH_LINE_INDEX	INTEGER	Each plan path in QUERY_PLAN_PROFILES could be represented with multiple rows. PATH_LINE_INDEX returns the relative line order. You should include the PATH_LINE_INDEX column in the QUERY_PLAN_PROFILES ... ORDER BY clause so rows in the result set appear as they do in EXPLAIN-generated query plans.
PATH_IS_EXECUTING	BOOLEAN	Status of a path in the query plan. True (t) if the path has started running, otherwise false.
PATH_IS_COMPLETE	BOOLEAN	Status of a path in the query plan. True (t) if the path has finished running, otherwise false.

Column Name	Data Type	Description
IS_EXECUTING	BOOLEAN	Status of a running query. True if the query is currently active (<i>t</i>), otherwise false (<i>f</i>).
RUNNING_TIME	INTERVAL	The amount of elapsed time the query path took to execute.
MEMORY_ALLOCATED_BYTES	INTEGER	The amount of memory the path used, in bytes.
READ_FROM_DISK_BYTES	INTEGER	The number of bytes the path read from disk (or the disk cache).
RECEIVED_BYTES	INTEGER	The number of bytes received over the network.
SENT_BYTES	INTEGER	Size of data sent over the network by the path.
PATH_LINE	VARCHAR	The query plan text string for the path, associated with the PATH ID and PATH_LINE_INDEX columns.

Privileges

No explicit permissions are required; however, users see only the records that correspond to tables they have permissions to view.

Best Practices

Table results can be very wide. For best results when you query the QUERY_PLAN_PROFILES table, sort on these columns:

- TRANSACTION_ID
- STATEMENT_ID
- PATH_ID
- PATH_LINE_INDEX

For example:

```
=> SELECT ... FROM query_plan_profiles  
    WHERE ...  
    ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

Example

See [Profiling Query Plans](#) in the Administrator's Guide

See Also

- [EXECUTION_ENGINE_PROFILES](#)
- [EXPLAIN](#)
- [PROFILE](#)
- [QUERY_EVENTS](#)

QUERY_PROFILES

Provides information about executed queries.

Column Name	Data Type	Description
SESSION_ID	VARCHAR	The identification of the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	An identifier for the transaction within the session if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.
IDENTIFIER	VARCHAR	A string to identify the query in system tables. Note: You can query the IDENTIFIER column to quickly identify queries you have labeled for profiling and debugging. See Labeling Queries in the Administrator's Guide for details.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
QUERY	VARCHAR	The query string used for the query.
QUERY_SEARCH_PATH	VARCHAR	A list of schemas in which to look for tables.
SCHEMA_NAME	VARCHAR	The schema name in which the query is being profiled.
TABLE_NAME	VARCHAR	The table name in the query being profiled.
QUERY_DURATION_US	NUMERIC (18,0)	The duration of the query in microseconds.
QUERY_START_EPOCH	INTEGER	The epoch number at the start of the given query.
QUERY_START	VARCHAR	The Linux system time of query execution in a format that can be used as a DATE/TIME expression.
QUERY_TYPE	VARCHAR	Is one of INSERT, SELECT, UPDATE, DELETE, UTILITY, or UNKNOWN.
ERROR_CODE	INTEGER	The return error code for the query.
USER_NAME	VARCHAR	The name of the user who ran the query.
PROCESSED_ROW_COUNT	INTEGER	The number of rows returned by the query.
RESERVED_EXTRA_MEMORY_B	INTEGER	Shows how much unused memory (in bytes) remains that is reserved for a given query but is unassigned to a specific operator. This is the memory from which unbounded operators pull first. The MEMORY_INUSE_KB column in system table RESOURCE_ACQUISITIONS shows how much total memory was acquired for each query. If operators acquire all memory acquired for the query, the plan must request more memory from the Vertica resource manager.
IS_EXECUTING	BOOLEAN	Displays information about actively running

Column Name	Data Type	Description
		queries, regardless of whether profiling is enabled.

QUERY_REQUESTS

Returns information about user-issued query requests.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_NAME	VARCHAR	Name of the user who issued the query at the time Vertica recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
REQUEST_ID	INTEGER	Unique identifier of the query request in the user session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.
REQUEST_TYPE	VARCHAR	Type of the query request. Examples include, but are not limited to: <ul style="list-style-type: none"> • QUERY • DDL • LOAD • UTILITY

Column Name	Data Type	Description
		<ul style="list-style-type: none"> TRANSACTION PREPARE EXECUTE SET SHOW
REQUEST	VARCHAR	Query statement.
REQUEST_LABEL	VARCHAR	Label of the query, if available.
SEARCH_PATH	VARCHAR	Contents of the search path.
MEMORY_ACQUIRED_MB	FLOAT	Memory acquired by this query request in megabytes.
SUCCESS	BOOLEAN	Value returned if the query successfully executed.
ERROR_COUNT	INTEGER	Number of errors encountered in this query request (logged in ERROR_MESSAGES table).
START_TIMESTAMP	TIMESTAMPTZ	Beginning of history interval.
END_TIMESTAMP	TIMESTAMPTZ	End of history interval.
REQUEST_DURATION_MS	INTEGER	Length of time the query ran in milliseconds.
IS_EXECUTING	BOOLEAN	Distinguishes between actively-running (t) and completed (f) queries.

Privileges

No explicit privileges are required. You can only see the records for requests associated with your user name. Users with [SYSMONITOR Role](#) or [DBADMIN Role](#) can see all records from all users.

See Also

[QUERY_PROFILES](#)

REBALANCE_OPERATIONS

Contains information on historic and ongoing rebalance operations.

Column Name	Data Type	Description
OBJECT_TYPE	VARCHAR	The type of the object being rebalanced. Valid types are: <ul style="list-style-type: none"> Projection DFSfile
OBJECT_ID	INT	The ID of the object being rebalanced.
OBJECT_NAME	VARCHAR	The name of the object being rebalanced. Objects can be tables, projections, or other Vertica objects.
PATH_NAME	VARCHAR	The DFS path for unstructured data being rebalanced.
TABLE_NAME	VARCHAR	The name of the table being rebalanced. This value is NULL for DFS files.
TABLE_SCHEMA	VARCHAR	The schema of the table being rebalanced. This value is NULL for DFS files.
TRANSACTION_ID	INT	The identifier for the transaction within the session.
STATEMENT_ID	INT	The unique numeric ID for the currently-running statement.
NODE_NAME	VARCHAR	Name of the node that is rebalancing.
OPERATION_NAME	VARCHAR	Identifies the specific rebalance operation being performed, set to one of the following: <ul style="list-style-type: none"> Refresh projection, update <i>temporary projection name</i> name and ID to <i>master projection name</i>

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • Drop unsegmented replicas • Replicate DFS File • Refresh projection • Drop replaced or replacement projection, rename <i>temporary projection name</i> to <i>original projection name</i> • Update temp table segments • Prepare : moveout • Prepare : separate • Move storage containers
OPERATION_STATUS	VARCHAR	<p>Returns Running or an empty string to indicate 'not running.' One of the following values:</p> <ul style="list-style-type: none"> • START • COMPLETE • ABORT
IS_EXECUTING	BOOLEAN	When TRUE, the operation is currently running.
REBALANCE_METHOD	VARCHAR	<p>The method that Vertica is using to perform the rebalance. Valid methods are:</p> <ul style="list-style-type: none"> • REFRESH: New projections are created according to the new segmentation definition. Data is copied via a refresh plan from projections with the previous segmentation to the new segments. This method is only used if specifically requested (using START_REFRESH), by setting a configuration parameter, or if Elastic Cluster is disabled or if there is a change in desired k-safety.

Column Name	Data Type	Description
		<ul style="list-style-type: none"> REPLICATE: Unsegmented projection data is copied to new nodes and removed from ephemeral nodes. ELASTIC_CLUSTER: The segmentation of existing segmented projections is altered to adjust to a new cluster topology and data is redistributed accordingly.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
OPERATION_START_TIMESTAMP	TIMESTAMPZ	The time that the rebalance began.
OPERATION_END_TIMESTAMP	TIMESTAMPZ	The time that the rebalance ended. If the rebalance is ongoing, this value is NULL.
ELASTIC_CLUSTER_VERSION	INT	The Elastic Cluster has a version. Each time the cluster topology changes, this version increments.
IS_LATEST	BOOLEAN	True if this row pertains to the most recent rebalance activity.

Privileges

Superuser

REBALANCE_PROJECTION_STATUS

Maintain history on rebalance progress for relevant projections.

Column Name	Data Type	Description
PROJECTION_ID	INTEGER	Identifier of the projection that will be, was, or is being rebalanced.
PROJECTION_SCHEMA	VARCHAR	Schema of the projection that will be, was, or is being

Column Name	Data Type	Description
		rebalanced.
PROJECTION_NAME	VARCHAR	Name of the projection that will be, was, or is being rebalanced.
ANCHOR_TABLE_ID	INTEGER	Anchor table identifier of the projection that will be, was, or is being rebalanced.
ANCHOR_TABLE_NAME	VARCHAR	Anchor table name of the projection that will be, was, or is being rebalanced.
REBALANCE_METHOD	VARCHAR	<p>Method that was, is, or will be used to rebalance the projection. Possible values are:</p> <ul style="list-style-type: none"> • REFRESH: New projections are created according to the new segmentation definition. Data is copied via a refresh plan from projections with the previous segmentation to the new segments. This method is only used if specifically requested (using START_REFRESH), by setting a configuration parameter, or if Elastic Cluster is disabled or if there is a change in desired k-safety. • REPLICATE: Unsegmented projection data is copied to new nodes and removed from ephemeral nodes. • ELASTIC_CLUSTER: The segmentation of existing segmented projections is altered to adjust to a new cluster topology and data is redistributed accordingly.
DURATION_SEC	INTERVAL SEC	<p>Deprecated - populated by NULL.</p> <p>Length of time (seconds) rebalance has been working on this projection, including time to separate storage, if that work is required.</p>
SEPARATED_PERCENT	NUMERIC (5,2)	Percent of storage that has been separated for this projection.
TRANSFERRED_PERCENT	NUMERIC	Percent of storage that has been transferred, for this

Column Name	Data Type	Description
	(5,2)	projection.
SEPARATED_BYTES	INTEGER	Number of bytes, separated by the corresponding rebalance operation, for this projection.
TO_SEPARATE_BYTES	INTEGER	Number of bytes that remain to be separated by the corresponding rebalance operation for this projection.
TRANSFERRED_BYTES	INTEGER	Number of bytes transferred by the corresponding rebalance operation for this projection.
TO_TRANSFER_BYTES	INTEGER	Number of bytes that remain to be transferred by the corresponding rebalance operation for this projection.
IS_LATEST	BOOLEAN	True if this row pertains to the most recent rebalance activity, where <code>elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster);</code>
ELASTIC_CLUSTER_VERSION	INTEGER	<p>The Elastic Cluster has a version, and each time the cluster topology changes, this version is incremented. This column reflects the version to which this row of information pertains. The TO_* fields (TO_SEPARATE_* and TO_TRANSFER_*) are only valid for the current version.</p> <p>To view only rows from the current, latest or upcoming rebalance operation, use:</p> <pre>WHERE elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster);</pre>

Privileges

Superuser

See Also

- [ELASTIC_CLUSTER](#)
- [REBALANCE_TABLE_STATUS](#)

REBALANCE_TABLE_STATUS

Maintain history on rebalance progress for relevant tables.

Column Name	Data Type	Description
TABLE_ID	INTEGER	Identifier of the table that will be, was, or is being rebalanced.
TABLE_SCHEMA	VARCHAR	Schema of the table that will be, was, or is being rebalanced.
TABLE_NAME	VARCHAR	Name of the table that will be, was, or is being rebalanced.
REBALANCE_METHOD	VARCHAR	Method that will be, is, or was used to rebalance the projections of this table. Possible values are: <ul style="list-style-type: none"> • REFRESH • REPLICATE • ELASTIC_CLUSTER
DURATION_SEC	INTERVAL SEC	Deprecated - populated by NULL. Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS .
SEPARATED_PERCENT	NUMERIC(5,2)	Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS .
TRANSFERRED_PERCENT	NUMERIC(5,2)	Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in

Column Name	Data Type	Description
		REBALANCE_PROJECTION_STATUS.
SEPARATED_BYTES	INTEGER	Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.
TO_SEPARATE_BYTES	INTEGER	Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.
TRANSFERRED_BYTES	INTEGER	Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.
TO_TRANSFER_BYTES	INTEGER	Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.
IS_LATEST	BOOLEAN	True if this row pertains to the most recent rebalance activity, where elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster;)
ELASTIC_CLUSTER_VERSION	INTEGER	<p>The Elastic Cluster has a version, and each time the cluster topology changes, this version is incremented. This column reflects the version to which this row of information pertains. The TO_* fields (TO_SEPARATE_* and TO_TRANSFER_*) are only valid for the current version.</p> <p>To view only rows from the current, latest or upcoming rebalance operation, use:</p> <pre>WHERE elastic_cluster_version = (SELECT version FROM v_ catalog.elastic_cluster;)</pre>
start_timestamp	TIMESTAMPZ	The time that the rebalance began.
end_timestamp	TIMESTAMPZ	The time that the rebalance ended.

Privileges

Superuser

See Also

- [ELASTIC_CLUSTER](#)
- [REBALANCE_PROJECTION_STATUS](#)

RECOVERY_STATUS

Provides the status of recovery operations, returning one row for each node.

Note: You cannot query this or other system tables table during cluster recovery; the cluster must be UP to accept connections.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
RECOVER_EPOCH	INTEGER	Epoch the recovery operation is trying to catch up to.
RECOVERY_PHASE	VARCHAR	Current stage in the recovery process. Can be one of the following: <ul style="list-style-type: none">• NULL• current• historical pass <i>X</i>, where <i>X</i> is the iteration count
SPLITS_COMPLETED	INTEGER	Number of independent recovery SPLITS queries that have run and need to run.
SPLITS_TOTAL	INTEGER	Total number of SPLITS queries that ran. Each query corresponds to one row in the PROJECTION_RECOVERIES table. If SPLITS_TOTAL = 2, then there

Column Name	Data Type	Description
		should be 2 rows added to PROJECTION_RECOVERIES, showing query details.
HISTORICAL_COMPLETED	INTEGER	Number of independent recovery HISTORICAL queries that have run and need to run.
HISTORICAL_TOTAL	INTEGER	Total number of HISTORICAL queries that ran. Each query corresponds to one row in the PROJECTION_RECOVERIES table. If HISTORICAL_TOTAL = 2, then there should be 2 rows added to PROJECTION_RECOVERIES, showing query details.
CURRENT_COMPLETED	INTEGER	Number of independent recovery CURRENT queries that have run and need to run.
CURRENT_TOTAL	INTEGER	Total number of CURRENT queries that ran. Each query corresponds to one row in the PROJECTION_RECOVERIES table. If CURRENT_TOTAL = 2, then there should be 2 rows added to PROJECTION_RECOVERIES, showing query details.
IS_RUNNING	BOOLEAN	True (<i>t</i>) if the node is still running recovery; otherwise false (<i>f</i>).

Privileges

None

See Also

[PROJECTION_RECOVERIES](#)

REPARENTED_ON_DROP

Lists re-parenting events of objects dropped from their original owner but still remain in Vertica. For example, a user may leave the organization and needs to be removed from the database. When the dbadmin drops the user from the database that user's objects are re-parented to another user.

In some cases a Vertica user's objects are reassigned based on the GlobalHeirUserName parameter. In this case a user's objects are re-parented to the user indicated by this parameter.

Column Name	Data Type	Description
REPARENT_TIMESTAMP	TIMESTAMP	The time the re-parenting event occurred.
NODE_NAME	VARCHAR	The name of the node or nodes on which the re-parenting occurred.
SESSION_ID	VARCHAR	The identification number of the re-parenting event.
USER_ID	INTEGER	The unique, system-generated user identification number.
USER_NAME	VARCHAR	The name of the user that caused the re-parenting event. For example, a dbadmin user may have dropped a user thus re-parenting that user's objects.
TRANSACTION_ID	INTEGER	The system-generated transaction identification number. Is NULL if a transaction id does not exist.
OLD_OWNER_NAME	VARCHAR	The the name of the dropped user who used to own the re-parented object.
OLD_OWNER_OID	INTEGER	The unique identification number of the user who used to own the re-parented object.
NEW_OWNER_NAME	VARCHAR	The name of the user who now owns the re-parented objects.
NEW_OWNER_OID	INTEGER	The unique identification number of the user who now owns the re-parented objects.
OBJ_NAME	VARCHAR	The name of the object being re-parented.
OBJ_OID	INTEGER	The unique identification number of the object being re-parented.
SCHEMA_NAME	VARCHAR	The name of the schema in which the object resides.
SCHEMA_OID	INTEGER	The unique identification number of the schema in which the re-parented object resides.

RESOURCE_ACQUISITIONS

Retains information about resources (memory, open file handles, threads) acquired by each running request for each resource pool in the system.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Node name for which information is listed.
TRANSACTION_ID	INTEGER	Transaction identifier for this request.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.
REQUEST_TYPE	VARCHAR	Type of request issued to a resource pool. Request type can be one of: <ul style="list-style-type: none">• Reserve: related to queries• Acquire: [Internal] related to the optimizer and other internal services, such as the Database Designer• Acquire additional: [Internal] related to size adjustment of acquisitions obtained through the first two methods; unusual, outside the WOS
POOL_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the resource pool. The ID represents the initial pool, even if the request has been moved to a new pool due to cascade. For more information about cascade events, see RESOURCE_POOL_MOVE .
POOL_NAME	VARCHAR	Name of the resource pool. The name represents the initial pool, even if the request has been moved to a new pool due to

Column Name	Data Type	Description
		cascade. For more information about cascade events, see RESOURCE_POOL_MOVE .
THREAD_COUNT	INTEGER	Number of threads in use by this request.
OPEN_FILE_HANDLE_COUNT	INTEGER	Number of open file handles in use by this request.
MEMORY_INUSE_KB	INTEGER	<p>Total amount of memory in kilobytes acquired by this query.</p> <p>The RESERVED_EXTRA_MEMORY_B column in system table QUERY_PROFILES shows how much unused memory (in bytes) remains that is reserved for a given query but is unassigned to a specific operator.</p> <p>If operators for a query acquire all memory specified by MEMORY_INUSE_KB, the plan must request more memory from the Vertica resource manager.</p>
QUEUE_ENTRY_TIMESTAMP	TIMESTAMPTZ	Timestamp when the request was queued at the Resource Manager.
ACQUISITION_TIMESTAMP	TIMESTAMPTZ	Timestamp when the request was admitted to run. See the Notes section below for the difference between these two timestamps.
RELEASE_TIMESTAMP	TIMESTAMPTZ	Time when Vertica released this resource acquisition.
DURATION_MS	INTEGER	Duration of the resource request in milliseconds.
IS_EXECUTING	BOOLEAN	Denotes if the query holding the resource is still executing (t).

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Notes

When monitoring resource pools and resource usage by queries, the “queue wait” time is the difference between `ACQUISITION_TIMESTAMP` and `QUEUE_ENTRY_TIMESTAMP`. For example, to determine how long a query waits in the queue before it is admitted to run, you can get the difference between the `ACQUISITION_TIMESTAMP` and the `QUEUE_ENTRY_TIMESTAMP` using a query like the following:

```
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,  
       (acquisition_timestamp-queue_entry_timestamp) AS 'queue wait'  
FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
```

See Also

- [QUERY_PROFILES](#)
- [RESOURCE_POOL_STATUS](#)
- [RESOURCE_POOLS](#)
- [RESOURCE_QUEUES](#)
- [RESOURCE_REJECTIONS](#)

RESOURCE_POOL_MOVE

Displays the cascade event information on each node.

Column Name	Data Type	Description
<code>NODE_NAME</code>	<code>VARCHAR</code>	Node name for which information is listed.
<code>MOVE_TIMESTAMP</code>	<code>TIMESTAMPZ</code>	Time when the query attempted to move to the target pool.
<code>SESSION_ID</code>	<code>VARCHAR</code>	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
<code>USER_ID</code>	<code>INTEGER</code>	Identifies the query event user.

Column Name	Data Type	Description
USER_NAME	VARCHAR	Name of the user for which Vertica lists query information at the time it records the session.
TRANSACTION_ID	INTEGER	Transaction identifier for the request.
STATEMENT_ID	INTEGER	Unique numeric ID for the statement.
SOURCE_POOL_NAME	VARCHAR	Name of the resource pool where the query was executing when Vertica attempted the move.
TARGET_POOL_NAME	VARCHAR	Name of resource pool where the query attempted to move.
MOVE_CAUSE	VARCHAR	Denotes why the query attempted to move. Valid values: <ul style="list-style-type: none"> • MOVE RESOURCE POOL COMMAND • RUNTIMECAP EXCEEDED
SOURCE_CAP	INTEGER	Effective RUNTIMECAP value for the source pool. The value represents the lowest of these three values: <ul style="list-style-type: none"> • session RUNTIMECAP • user RUNTIMECAP • source pool RUNTIMECAP
TARGET_CAP	INTEGER	Effective RUNTIMECAP value for the target pool. The value represents the lowest of these three values: <ul style="list-style-type: none"> • session RUNTIMECAP • user RUNTIMECAP • target pool RUNTIMECAP
SUCCESS	BOOLEAN	True, if the query successfully moved to the target pool.
RESULT_REASON	VARCHAR	States reason for success or failure of the move.

See Also

- [QUERY_PROFILES](#)
- [RESOURCE_POOL_STATUS](#)
- [RESOURCE_POOLS](#)
- [RESOURCE_QUEUES](#)
- [RESOURCE_REJECTIONS](#)

RESOURCE_POOL_STATUS

Provides configuration settings of the various resource pools in the system, including internal pools. For detailed information about resource parameters, see [CREATE RESOURCE POOL](#) or [ALTER RESOURCE POOL](#).

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node for which information is provided.
POOL_OID	INTEGER	A unique numeric ID that identifies the pool and is assigned by the Vertica catalog.
POOL_NAME	VARCHAR	The name of the resource pool.
IS_INTERNAL	BOOLEAN	Denotes whether a pool is one of the Built-In Pools .
MEMORY_SIZE_KB	INTEGER	Value of MEMORYSIZE setting of the pool in kilobytes.
MEMORY_SIZE_ACTUAL_KB	INTEGER	Current amount of memory, in kilobytes, allocated to the pool by the resource manager. The actual size can be less than specified in the DDL, if both the following conditions exist: <ul style="list-style-type: none">• The pool has been recently altered in a running system.• The request to shuffle memory is pending.

Column Name	Data Type	Description
MEMORY_INUSE_KB	INTEGER	Amount of memory, in kilobytes, acquired by requests running against this pool.
GENERAL_MEMORY_BORROWED_KB	INTEGER	Amount of memory, in kilobytes, borrowed from the GENERAL pool by requests running against this pool. The sum of MEMORY_INUSE_KB and GENERAL_MEMORY_BORROWED_KB should be less than MAX_MEMORY_SIZE_KB.
QUEUEING_THRESHOLD_KB	INTEGER	Calculated as $MAX_MEMORY_SIZE_KB * 0.95$. When the amount of memory used by all requests against this resource pool exceeds the QUEUEING_THRESHOLD_KB, new requests against the pool are queued until memory becomes available.
MAX_MEMORY_SIZE_KB	INTEGER	<p>Value, in kilobytes, of the MAXMEMORYSIZE parameter specified when defining the pool. After this threshold is reached, new requests against this pool are rejected or queued until memory becomes available.</p> <p>MAX_MEMORY_SIZE_KB might not reflect the set MAXMEMORYSIZE parameter value if the specified value cannot be reached.</p> <p>Example:</p> <p>If MAXMEMORYSIZE = 10G, but less than 2G is available, MAX_MEMORY_SIZE_KB will not reflect the original value in KB, but will instead reflect 2G in KB, since that is the highest value that can be reached.</p> <p>Ordinarily, the ALTER RESOURCE POOL and CREATE RESOURCE POOL statements provide error messages to prevent setting the MAXMEMORYSIZE to a value greater than available, but making system changes or other configuration changes can cause this scenario.</p>
RUNNING_QUERY_COUNT	INTEGER	Number of queries actually running using this pool.
PLANNED_CONCURRENCY	INTEGER	Value of PLANNEDCONCURRENCY parameter specified when defining the pool.

Column Name	Data Type	Description
MAX_CONCURRENCY	INTEGER	Value of MAXCONCURRENCY parameter specified when defining the pool.
IS_STANDALONE	BOOLEAN	If the pool is configured to have MEMORYSIZE equal to MAXMEMORYSIZE, the pool is considered standalone because it does not borrow any memory from the General pool.
QUEUE_TIMEOUT	INTERVAL	The interval that the request waits for resources to become available before being rejected. If you set this value to NONE, Vertica displays it as NULL.
QUEUE_TIMEOUT_IN_SECONDS	INTEGER	Value of QUEUETIMEOUT parameter that was specified when defining the pool. If you set QUEUE_TIMEOUT to NONE, Vertica displays this value as NULL.
EXECUTION_PARALLELISM	INTEGER	Limits the number of threads used to process any single query issued in this resource pool.
PRIORITY	INTEGER	Value of PRIORITY parameter specified when defining the pool. When set to HOLD, Vertica sets a pool's priority to -999 so the query remains queued until QUEUETIMEOUT is reached.
RUNTIME_PRIORITY	VARCHAR	Value of RUNTIME_PRIORITY specified when defining the pool.
RUNTIME_PRIORITY_THRESHOLD	INTEGER	Value of RUNTIME_PRIORITY_THRESHOLD specified when defining the pool.
SINGLE_INITIATOR	BOOLEAN	Value of SINGLEINITIATOR parameter specified when defining the pool.
QUERY_BUDGET_KB	INTEGER	The current amount of memory that queries are tuned to use. The calculation that Vertica uses to determine this value is described in Target Memory Determination for Queries in Concurrent Environments . Note: The calculated value can change when one or

Column Name	Data Type	Description
		<p>more running queries needs more than the budgeted amount to run.</p> <p>For a detailed example of query budget calculations, see Do You Need to Put Your Query on a Budget? in the Vertica User Community.</p>
CPU_AFFINITY_SET	VARCHAR	<p>The set of CPUs on which queries associated with this pool are executed. Can be:</p> <ul style="list-style-type: none"> • A percentage of CPUs on the system • A zero-based list of CPUs (a four-CPU system c of CPUs 0, 1, 2, and 3).
CPU_AFFINITY_MASK	VARCHAR	<p>The bit mask of CPUs available for use in this pool.</p> <p>Valid values: Hex-encoded binaries for the CPUs on the system. Read from right to left.</p> <p>Example:</p> <p>A pool that has exclusive use of CPUs 0 and 1 on a four-CPU system is represented by the hex value of 3, which translates to 0011. The 1's indicates the that CPU 0 and 1 are available.</p> <p>If no other resource pools use exclusive CPU affinities, the remaining resource pools in this example have a mask of 3, which translates to 1100. Thus, CPUs 2 and 3 of the four CPUs on the system are available to these resource pools.</p>
CPU_AFFINITY_MODE	VARCHAR	<p>The mode for the CPU affinity, one of the following:</p> <ul style="list-style-type: none"> • ANY • EXCLUSIVE • SHARED

See Also

- [CREATE RESOURCE POOL](#)
- [RESOURCE_ACQUISITIONS](#)
- [RESOURCE_POOLS](#)
- [RESOURCE_QUEUES](#)
- [RESOURCE_REJECTIONS](#)
- [Managing Workloads](#)
- [Using Queries to Monitor Resource Pool Size and Usage](#)

RESOURCE_QUEUES

Provides information about requests pending for various resource pools.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The name of the node for which information is listed.
TRANSACTION_ID	INTEGER	Transaction identifier for this request
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.
POOL_NAME	VARCHAR	The name of the resource pool
MEMORY_REQUESTED_KB	INTEGER	Amount of memory in kilobytes requested by this request
PRIORITY	INTEGER	Value of PRIORITY parameter specified when defining the pool.

Column Name	Data Type	Description
POSITION_IN_QUEUE	INTEGER	Position of this request within the pool's queue
QUEUE_ENTRY_TIMESTAMP	TIMESTAMP	Timestamp when the request was queued

See Also

- [RESOURCE_ACQUISITIONS](#)
- [RESOURCE_POOLS](#)
- [RESOURCE_REJECTIONS](#)

RESOURCE_REJECTION_DETAILS

Records an entry for each resource request that Vertica denies. This is useful for determining if there are resource space issues, as well as which users/pools encounter problems.

Column Name	Data Type	Description
REJECTED_TIMESTAMP	TIMESTAMPTZ	Time when Vertica rejected the resource.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_NAME	VARCHAR	Name of the user at the time Vertica recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
REQUEST_ID	INTEGER	Unique identifier of the query request in the user session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
STATEMENT_ID	INTEGER	Unique numeric ID for the currently-running statement. NULL indicates that no statement is

Column Name	Data Type	Description
		currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.
POOL_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the resource pool.
POOL_NAME	VARCHAR	Name of the resource pool
REASON	VARCHAR	Reason for rejecting this request; for example: <ul style="list-style-type: none"> • Usage of single request exceeds high limit • Timed out waiting for resource reservation • Canceled waiting for resource reservation
RESOURCE_TYPE	VARCHAR	Memory, threads, file handles or execution slots. The following list shows the resources that are limited by the resource manager. A query might need some amount of each resource, and if the amount needed is not available, the query is queued and could eventually time out of the queue and be rejected. <ul style="list-style-type: none"> • Number of running plans • Number of running plans on initiator node (local) • Number of requested threads • Number of requested file handles • Number of requested KB of memory • Number of requested KB of address space <p>Note: Execution slots are determined by MAXCONCURRENCY parameter.</p>
REJECTED_VALUE	INTEGER	Amount of the specific resource requested by the last rejection

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

[RESOURCE_REJECTIONS](#)

RESOURCE_REJECTIONS

Monitors requests for resources that are rejected by the Resource Manager. Information is valid only as long as the node is up and the counters reset to 0 upon node restart.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
POOL_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the resource pool.
POOL_NAME	VARCHAR	The name of the resource pool.
REASON	VARCHAR	The reason for rejecting this request; for example: <ul style="list-style-type: none">• Usage of single request exceeds high limit• Timed out waiting for resource reservation• Canceled waiting for resource reservation
RESOURCE_TYPE	VARCHAR	Memory, threads, file handles or execution slots. The following list shows the resources that are limited by the resource manager. A query might need some amount of each resource, and if the amount needed is not available, the query is queued and could eventually time out of the queue and be rejected.

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • Number of running plans • Number of running plans on initiator node (local) • Number of requested threads • Number of requested file handles • Number of requested KB of memory • Number of requested KB of address space <p>Note: Execution slots are determined by MAXCONCURRENCY parameter.</p>
REJECTION_COUNT	INTEGER	Number of requests rejected due to specified reason and RESOURCE_TYPE.
FIRST_REJECTED_TIMESTAMP	TIMESTAMPTZ	The time of the first rejection for this pool Reasons for Rejection <ul style="list-style-type: none"> • Usage of single request exceeds high limit • Timed out waiting for resource reservation • Canceled waiting for resource reservation
LAST_REJECTED_TIMESTAMP	TIMESTAMPTZ	The time of the last rejection for this pool
LAST_REJECTED_VALUE	INTEGER	The amount of the specific resource requested by the last rejection

Example

```
=> SELECT node_name, pool_name, reason, resource_type,
       rejection_count AS count,
       last_rejected_value AS value
       FROM resource_rejections;
node_name | pool_name |          reason          | resource_type | count |
value
-----+-----+-----+-----+-----+-----
```

```

-----
initiator | alsohassome | Request exceeded high limit          | Memory(KB)      | 1 |
102400
initiator | ceo          | Timedout waiting for resource request | Memory(KB)      | 1 |
102400
initiator | empty        | Request exceeded high limit          | Queries         | 1 |
1
initiator | general      | Request exceeded high limit          | Address space(KB) | 2 |
45035996273704970
initiator | general      | Request exceeded high limit          | Memory(KB)      | 24 |
8395584
initiator | sa           | Request exceeded high limit          | Memory(KB)      | 3 |
102400
initiator | sa           | Timedout waiting for resource request | Memory(KB)      | 1 |
10
initiator | small        | Request exceeded high limit          | Memory(KB)      | 26 |
102400
initiator | small        | Timedout waiting for resource request | Memory(KB)      | 2 |
122880
initiator | sysdata     | Request exceeded high limit          | Memory(KB)      | 5 |
102400
(10 rows)

```

The following command returns the type of resources currently running on the node:

```

=> SELECT resource_type FROM resource_rejections;
resource_type
-----
UPDATE_QUERY
UPDATE_QUERY
UPDATE_QUERY
(3 rows)

```

See Also

- [CLEAR_RESOURCE_REJECTIONS](#)
- [DISK_RESOURCE_REJECTIONS](#)

RESOURCE_USAGE

Monitors system resource management on each node.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.

Column Name	Data Type	Description
REQUEST_COUNT	INTEGER	The cumulative number of requests for threads, file handles, and memory (in kilobytes).
LOCAL_REQUEST_COUNT	INTEGER	The cumulative number of local requests.
REQUEST_QUEUE_DEPTH	INTEGER	The current request queue depth.
ACTIVE_THREAD_COUNT	INTEGER	The current number of active threads.
OPEN_FILE_HANDLE_COUNT	INTEGER	The current number of open file handles.
MEMORY_REQUESTED_KB	INTEGER	The memory requested in kilobytes.
ADDRESS_SPACE_REQUESTED_KB	INTEGER	The address space requested in kilobytes.
WOS_USED_BYTES	INTEGER	The size of the WOS in bytes.
WOS_ROW_COUNT	INTEGER	The number of rows in the WOS.
ROS_USED_BYTES	INTEGER	The size of the ROS in bytes.
ROS_ROW_COUNT	INTEGER	The number of rows in the ROS.
TOTAL_USED_BYTES	INTEGER	The total size of storage (WOS + ROS) in bytes.
TOTAL_ROW_COUNT	INTEGER	The total number of rows in storage (WOS + ROS).
RESOURCE_REQUEST_REJECT_COUNT	INTEGER	The number of rejected plan requests.
RESOURCE_REQUEST_TIMEOUT_COUNT	INTEGER	The number of resource request timeouts.
RESOURCE_REQUEST_CANCEL_COUNT	INTEGER	The number of resource request cancelations.
DISK_SPACE_REQUEST_REJECT_COUNT	INTEGER	The number of rejected disk write requests.
FAILED_VOLUME_REJECT_COUNT	INTEGER	The number of rejections due to a failed volume.
TOKENS_USED	INTEGER	For internal use only.
TOKENS_AVAILABLE	INTEGER	For internal use only.

SESSION_MARS_STORE

Shows Multiple Active Result Sets (MARS) storage information.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
SESSION_ID	VARCHAR	Identifier of the Vertica session. This identifier is unique within the cluster for the current session but can be reused in a subsequent session.
USER_NAME	VARCHAR	The username used to create the connection.
RESULTSET_ID	INT	Identifier assigned to the result set.
ROW_COUNT	INT	Number of rows requested by the query.
REMAINING_ROW_COUNT	INT	Number of rows that still need to be returned.
BYTES_USED	INT	The number of bytes requested.

SESSION_PARAMETERS

Provides information about current parameters that are configurable at the session level. To view parameters configurable at all levels, see the [CONFIGURATION_PARAMETERS](#) system table.

Column Name	Data Type	Description
SESSION_ID	VARCHAR	The unique identifier for the session.
SCHEMA_NAME	VARCHAR	The name of the schema on which the session is running.
LIB_NAME	VARCHAR	The name of the user library running the UDx, if necessary.
LIB_OID	VARCHAR	The object ID of the library containing the function, if one is running.
PARAMETER_NAME	VARCHAR	The name of the session parameter.

Column Name	Data Type	Description
CURRENT_VALUE	VARCHAR	The value of the session parameter.

See Also

[Configuration Parameters](#)

SESSION_PROFILES

Provides basic session parameters and lock time out data. To obtain information about sessions, see [Profiling Database Performance](#).

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
USER_NAME	VARCHAR	The name used to log in to the database or NULL if the session is internal.
CLIENT_HOSTNAME	VARCHAR	The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal.
LOGIN_TIMESTAMP	TIMESTAMP	The date and time the user logged into the database or when the internal session was created. This field is useful for identifying sessions that have been left open for a period of time and could be idle.
LOGOUT_TIMESTAMP	TIMESTAMP	The date and time the user logged out of the database or when the internal session was closed.
SESSION_ID	VARCHAR	A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.

Column Name	Data Type	Description
EXECUTED_STATEMENT_SUCCESS_COUNT	INTEGER	The number of successfully run statements.
EXECUTED_STATEMENT_FAILURE_COUNT	INTEGER	The number of unsuccessfully run statements.
LOCK_GRANT_COUNT	INTEGER	The number of locks granted during the session.
DEADLOCK_COUNT	INTEGER	The number of deadlocks encountered during the session.
LOCK_TIMEOUT_COUNT	INTEGER	The number of times a lock timed out during the session.
LOCK_CANCELLATION_COUNT	INTEGER	The number of times a lock was canceled during the session.
LOCK_REJECTION_COUNT	INTEGER	The number of times a lock was rejected during a session.
LOCK_ERROR_COUNT	INTEGER	The number of lock errors encountered during the session.
CLIENT_TYPE	VARCHAR	The type of client from which the connection was made. Possible client type values: <ul style="list-style-type: none"> • ADO.NET Driver • ODBC Driver • JDBC Driver • vsql
CLIENT_VERSION	VARCHAR	Returns the client version.
CLIENT_OS	VARCHAR	Returns the client operating system.
CLIENT_OS_USER_NAME	VARCHAR	The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.

See Also

[LOCKS](#)

SESSIONS

Monitors external sessions. Use this table to perform the following tasks:

- Identify users who are running lengthy queries.
- Identify users who hold locks because of an idle but uncommitted transaction.
- Determine the details of the database security used for a particular session, either Secure Socket Layer (SSL) or client authentication.
- Identify client-specific information, such as client version.

Note: During session initialization and termination, you might see sessions running only on nodes other than the node on which you ran the virtual table query. This is a temporary situation that corrects itself when session initialization and termination complete.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
USER_NAME	VARCHAR	The name used to log in to the database or NULL if the session is internal.
CLIENT_HOSTNAME	VARCHAR	The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal. Vertica accepts either IPv4 or IPv6 connections from a client machine. If the client machine contains mappings for both IPv4 and IPv6, the server randomly chooses one IP address family to make a connection. This can cause the CLIENT_HOSTNAME column to display either IPv4 or IPv6 values, based on which address family the server chooses.

Column Name	Data Type	Description
CLIENT_PID	INTEGER	The process identifier of the client process that issued this connection. Remember that the client process could be on a different machine than the server.
LOGIN_TIMESTAMP	TIMESTAMP	The date and time the user logged into the database or when the internal session was created. This field can help you identify sessions that have been left open for a period of time and could be idle.
SESSION_ID	VARCHAR	The identifier required to close or interrupt a session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
IDLE_SESSION_TIMEOUT	VARCHAR	Specifies how long this session can remain idle before timing out, set by SET SESSION IDLESESSIONTIMEOUT .
GRACE_PERIOD	VARCHAR	Specifies how long a session socket remains blocked while awaiting client input or output for a given query, set by SET SESSION GRACEPERIOD . If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated.
CLIENT_LABEL	VARCHAR	A user-specified label for the client connection that can be set when using ODBC. See Label in Data Source Name (DSN) Connection Properties in Connecting to Vertica . An MC output value means there are is a client connection to an MC-managed database for that USER_NAME
TRANSACTION_START	DATE	The date/time the current transaction started or NULL if no transaction is running.
TRANSACTION_ID	INTEGER	A string containing the hexadecimal representation of the transaction ID, if any; otherwise, NULL.

Column Name	Data Type	Description
TRANSACTION _ DESCRIPTION	VARCHAR	Description of the current transaction.
STATEMENT_START	TIMESTAMP	The timestamp the current statement started execution, or NULL if no statement is running.
STATEMENT_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the currently-executing statement. A value of NULL indicates that no statement is currently being processed.
LAST_STATEMENT_ DURATION_US	INTEGER	The duration of the last completed statement in microseconds.
RUNTIME_PRIORITY	VARCHAR	Specifies how many run-time resources (CPU, I/O bandwidth) are allocated to queries that are running in the resource pool.
CURRENT_ STATEMENT	VARCHAR	The currently executing statement, if any. NULL indicates that no statement is currently being processed.
LAST_STATEMENT	VARCHAR	NULL if the user has just logged in; otherwise the currently running statement or the most recently completed statement.
SSL_STATE	VARCHAR	Indicates if Vertica used Secure Socket Layer (SSL) for a particular session. Possible values are: <ul style="list-style-type: none"> • None—Vertica did not use SSL. • Server—Server authentication was used, so the client could authenticate the server. • Mutual—Both the server and the client authenticated one another through mutual authentication. See Security and Authentication and TLS/SSL Server Authentication .

Column Name	Data Type	Description
AUTHENTICATION_METHOD	VARCHAR	<p>The type of client authentication used for a particular session, if known. Possible values are:</p> <ul style="list-style-type: none"> • Unknown • Trust • Reject • Hash • Ident • LDAP • GSS • TLS <p>See Security and Authentication and Implementing Client Authentication.</p>
CLIENT_TYPE	VARCHAR	<p>The type of client from which the connection was made. Possible client type values:</p> <ul style="list-style-type: none"> • ADO.NET Driver • ODBC Driver • JDBC Driver • vsql
CLIENT_VERSION	VARCHAR	Client version.
CLIENT_OS	VARCHAR	Client operating system.
CLIENT_OS_USER_NAME	VARCHAR	The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.
CLIENT_AUTHENTICATION_NAME	VARCHAR	User-assigned name of the authentication method.

Column Name	Data Type	Description
CLIENT_AUTHENTICATION	INTEGER	Object identifier of the client authentication method.
REQUESTED_PROTOCOL	INTEGER	The requested protocol to be used when connecting.
EFFECTIVE_PROTOCOL	INTEGER	The protocol used when connecting.
EXTERNAL_MEMORY_KB	INTEGER	Amount of memory consumed by the Java Virtual Machines associated with the session.

Privileges

A superuser has unrestricted access to all session information. Users can view information only about their own, current sessions.

See Also

- [CLOSE_SESSION](#)
- [CLOSE_ALL_SESSIONS](#)

STORAGE_CONTAINERS

Monitors information about WOS and ROS storage containers in the database.

Column Name	Data Type	Description
NODE_NAME*	VARCHAR	Node name for which information is listed.
SCHEMA_NAME*	VARCHAR	Schema name for which information is listed.
PROJECTION_ID*	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_	VARCHAR	Projection name for which information is listed on that

* Column values cached for faster query performance

Column Name	Data Type	Description
NAME*		node.
STORAGE_TYPE*	VARCHAR	Type of storage container: ROS or WOS.
STORAGE_OID*	INTEGER	Numeric ID assigned by the Vertica catalog, which identifies the storage.
SAL_STORAGE_ID	VARCHAR	Unique hexadecimal numeric ID assigned by the Vertica catalog, which identifies the storage.
TOTAL_ROW_COUNT*	VARCHAR	Total rows in the storage container listed for that projection.
DELETED_ROW_COUNT*	INTEGER	Total rows in the storage container deleted for that projection.
USED_BYTES*	INTEGER	Total bytes in the storage container listed for that projection.
START_EPOCH*	INTEGER	Number of the start epoch in the storage container for which information is listed.
END_EPOCH*	INTEGER	Number of the end epoch in the storage container for which information is listed.
GROUPING	VARCHAR	The group by which columns are stored: <ul style="list-style-type: none"> • ALL: All columns are grouped • PROJECTION: Columns grouped according to projection definition • NONE: No columns grouped, despite grouping in the projection definition • OTHER: Some grouping but neither all nor according to projection (e.g., results from add column)
SEGMENT_LOWER_BOUND	INTEGER	Lower bound of the segment range spanned by the storage container or NULL if the corresponding projection is not elastic.
SEGMENT_UPPER_BOUND	INTEGER	Upper bound of the segment range spanned by the

* Column values cached for faster query performance, continued

Column Name	Data Type	Description
BOUND		storage container or NULL if the corresponding projection is not elastic.
IS_SORTED	BOOLEAN	Whether the storage container's data is sorted (WOS containers only).
LOCATION_LABEL	VARCHAR (128)	The location label (if any) for the storage container is stored.
DELETE_VECTOR_COUNT	INTEGER	The number of delete vectors in the storage container.

* Column values cached for faster query performance, continued

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

STORAGE_POLICIES

Monitors the current storage policies in effect for one or more database objects.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	Schema name for which information is listed.
OBJECT_NAME	VARCHAR	The name of the database object associated through the storage policy.
POLICY_DETAILS	VARCHAR	The object type of the storage policy.
LOCATION_LABEL	VARCHAR (128)	The label for this storage location.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

- [PARTITIONS](#)
- [STORAGE_CONTAINERS](#)
- [STORAGE_USAGE](#)

STORAGE_TIERS

Provides information about all storage locations with the same label across all cluster nodes. This table lists data totals for all same-name labeled locations.

The system table shows what labeled locations exist on the cluster, as well as other cluster-wide data about the locations.

Column Name	Data Type	Description
LOCATION_LABEL	VARCHAR	The label associated with a specific storage location. The <code>storage_tiers</code> system table includes data totals for unlabeled locations, which are considered labeled with empty strings (' ').
NODE_COUNT	INTEGER	The total number of nodes that include a storage location named <code>location_label</code> .
LOCATION_COUNT	INTEGER	The total number of storage locations named <code>location_label</code> . This value can differ from <code>node_count</code> if you create labeled locations with the same name at different paths on different nodes. For example: v_vmart_node0001: Create one labeled location, FAST V_vmart_node0002: Create two labeled locations, FAST, at different directory paths In this case, <code>node_count</code> value = 2, while <code>location_count</code> value = 3.
ROS_CONTAINER_COUNT	INTEGER	The total number of ROS containers stored across all cluster nodes for <code>location_label</code> .

Column Name	Data Type	Description
TOTAL_OCCUPIED_SIZE	INTEGER	The total number of bytes that all ROS containers for <code>location_label</code> occupy across all cluster nodes.

Privileges

None

See Also

- [DISK_STORAGE](#)
- [STORAGE_POLICIES](#)
- [STORAGE_USAGE](#)
- [Storage Management Functions](#)

STORAGE_USAGE

Provides information about file system storage usage. This is useful for determining disk space usage trends.

Column Name	Data Type	Description
POLL_TIMESTAMP	TIMESTAMPTZ	Time when Vertica recorded the row.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
PATH	VARCHAR	Path where the storage location is mounted.
DEVICE	VARCHAR	Device on which the storage location is mounted.
FILESYSTEM	VARCHAR	Filesystem on which the storage location is mounted.
USED_BYTES	INTEGER	Counter history of number of used bytes.
FREE_BYTES	INTEGER	Counter history of number of free bytes.

Column Name	Data Type	Description
USAGE_PERCENT	FLOAT	Percent of storage in use.

Privileges

Superuser

See Also

- [DISK_STORAGE](#)
- [STORAGE_CONTAINERS](#)
- [STORAGE_POLICIES](#)
- [STORAGE_TIERS](#)
- [Storage Management Functions](#)

STRATA

Contains internal details of how the Tuple Mover combines ROS containers in each projection, broken down by stratum and classifies the ROS containers by size and partition. The related [STRATA_STRUCTURES](#) table provides a summary of the strata values.

[Mergeout](#) in the Administrator's Guide describes how the Tuple Mover combines ROS containers.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed
SCHEMA_NAME	VARCHAR	The schema name for which information is listed
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed on that node

Column Name	Data Type	Description
PARTITION_KEY	VARCHAR	The data partition for which information is listed
STRATA_COUNT	INTEGER	The total number of strata for this projection partition
MERGING_STRATA_COUNT	INTEGER	The number of strata the Tuple Mover can merge out.
STRATUM_CAPACITY	INTEGER	The maximum number of ROS containers for the stratum before they must be merged.
STRATUM_HEIGHT	FLOAT	The size ratio between the smallest and largest ROS container in this stratum
STRATUM_NO	INTEGER	The stratum number. Strata are numbered starting at 0, for the stratum containing the smallest ROS containers
STRATUM_LOWER_SIZE	VARCHAR	The smallest ROS container size allowed in this stratum
STRATUM_UPPER_SIZE	VARCHAR	The largest ROS container size allowed in this stratum
ROS_CONTAINER_COUNT	INTEGER	The current number of ROS containers in the projection partition

STRATA_STRUCTURES

New column for VER-22441: projection_id

This table provides an overview of Tuple Mover internal details. It summarizes how the ROS containers are classified by size. A more detailed view can be found in the [STRATA](#) virtual table.

Column Name	Data Type	Description
ACTIVE_STRATA_COUNT	INTEGER	The total number of strata that have ROS containers in them
MERGING_STRATA_COUNT	INTEGER	In certain hardware configurations, a high strata could contain more ROS containers than the Tuple Mover can merge out; output from this column denotes the number of strata the Tuple Mover can merge out.
NODE_NAME	VARCHAR	The node name for which information is listed
PARTITION_KEY	VARCHAR	The data partition for which the information is listed

Column Name	Data Type	Description
PROJECTION_ID	INTEGER	A unique numeric ID assigned by the Vertica catalog, which identifies the projection.
PROJECTION_NAME	VARCHAR	The projection name for which information is listed on that node
SCHEMA_NAME	VARCHAR	The schema name for which information is listed
STRATUM_CAPACITY	INTEGER	The maximum number of ROS containers that the strata can contained before it must merge them
STRATA_COUNT	INTEGER	The total number of strata for this projection partition
STRATUM_HEIGHT	FLOAT	The size ratio between the smallest and largest ROS container in a stratum.

Example

```

onenode=> SELECT * FROM strata_structures;
-[ RECORD 1 ]-----+-----
node_name          | v_onenode_node0001
schema_name        | public
projection_name    | trades_p
projection_id      | 45035996273718838
partition_key      |
strata_count       | 11
merging_strata_count | 4
stratum_capacity   | 32
stratum_height     | 28.8
active_strata_count | 1
vmartdb=> \pset expanded
Expanded display is on.
vmartdb=> SELECT node_name, schema_name, projection_name, strata_count,
              stratum_capacity, stratum_height, active_strata_count
              FROM strata_structures WHERE stratum_capacity > 60;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
schema_name        | public
projection_name    | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key      |
strata_count       | 4
stratum_capacity   | 62
stratum_height     | 25.6511590887058
active_strata_count | 1
-[ RECORD 2 ]-----+-----
node_name          | v_vmartdb_node0001
schema_name        | public
projection_name    | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key      |

```

```

strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1
-[ RECORD 3 ]-----+-----
node_name         | v_vmartdb_node0002
schema_name       | public
projection_name   | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key     |
strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1
-[ RECORD 4 ]-----+-----
node_name         | v_vmartdb_node0002
schema_name       | public
projection_name   | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key     |
strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1
-[ RECORD 5 ]-----+-----
node_name         | v_vmartdb_node0003
schema_name       | public
projection_name   | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key     |
strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1
-[ RECORD 6 ]-----+-----
node_name         | v_vmartdb_node0003
schema_name       | public
projection_name   | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key     |
strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1
-[ RECORD 7 ]-----+-----
node_name         | v_vmartdb_node0004
schema_name       | public
projection_name   | shipping_dimension_DBD_22_seg_vmart_design_vmart_design
partition_key     |
strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1
-[ RECORD 8 ]-----+-----
node_name         | v_vmartdb_node0004
schema_name       | public
projection_name   | shipping_dimension_DBD_23_seg_vmart_design_vmart_design
partition_key     |
strata_count      | 4
stratum_capacity  | 62
stratum_height    | 25.6511590887058
active_strata_count | 1

```

See Also

- [STRATA](#)

SYSTEM

Monitors the overall state of the database.

Column Name	Data Type	Description
CURRENT_EPOCH	INTEGER	The current epoch number.
AHM_EPOCH	INTEGER	The AHM epoch number.
LAST_GOOD_EPOCH	INTEGER	The smallest (min) of all the checkpoint epochs on the cluster.
REFRESH_EPOCH	INTEGER	Deprecated, always set to -1.
DESIGNED_FAULT_TOLERANCE	INTEGER	The designed or intended K-safety level.
NODE_COUNT	INTEGER	The number of nodes in the cluster.
NODE_DOWN_COUNT	INTEGER	The number of nodes in the cluster that are currently down.
CURRENT_FAULT_TOLERANCE	INTEGER	The number of node failures the cluster can tolerate before it shuts down automatically. This is the current K-safety level.
CATALOG_REVISION_NUMBER	INTEGER	The catalog version number.
WOS_USED_BYTES	INTEGER	The WOS size in bytes (cluster-wide).
WOS_ROW_COUNT	INTEGER	The number of rows in WOS (cluster-wide).
ROS_USED_BYTES	INTEGER	The ROS size in bytes (cluster-wide).
ROS_ROW_COUNT	INTEGER	The number of rows in ROS (cluster-wide).
TOTAL_USED_	INTEGER	The total storage in bytes (WOS + ROS) (cluster-wide).

Column Name	Data Type	Description
BYTES		
TOTAL_ROW_COUNT	INTEGER	The total number of rows (WOS + ROS) (cluster-wide).

SYSTEM_RESOURCE_USAGE

Provides history about system resources, such as memory, CPU, network, disk, I/O.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
END_TIME	TIMESTAMP	End time of the history interval.
AVERAGE_MEMORY_USAGE_PERCENT	FLOAT	Average memory usage in percent of total memory (0-100) during the history interval.
AVERAGE_CPU_USAGE_PERCENT	FLOAT	Average CPU usage in percent of total CPU time (0-100) during the history interval.
NET_RX_KBYTES_PER_SECOND	FLOAT	Average number of kilobytes received from network (incoming) per second during the history interval.
NET_TX_KBYTES_PER_SECOND	FLOAT	Average number of kilobytes transmitting to network (outgoing) per second during the history interval.
IO_READ_KBYTES_PER_SECOND	FLOAT	Disk I/O average number of kilobytes read from disk per second during the history interval.
IO_WRITTEN_KBYTES_PER_SECOND	FLOAT	Average number of kilobytes written to disk per second during the history interval.

Privileges

Superuser

SYSTEM_SERVICES

Provides information about background system services that the Workload Analyzer monitors.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
SERVICE_TYPE	VARCHAR	Type of service; can be one of: <ul style="list-style-type: none">• SYSTEM• TUPLE MOVER
SERVICE_GROUP	VARCHAR	Group name, if there are multiple services of the same type.
SERVICE_NAME	VARCHAR	Name of the service.
SERVICE_INTERVAL_SEC	INTEGER	How often the service is executed (in seconds) during the history interval.
IS_ENABLED	BOOLEAN	Denotes if the service is enabled.
LAST_RUN_START	TIMESTAMPTZ	Denotes when the service was started last time.
LAST_RUN_END	TIMESTAMPTZ	Denotes when the service was completed last time.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

SYSTEM_SESSIONS

Provides information about system internal session history by system task.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_NAME	VARCHAR	Name of the user at the time Vertica recorded the session.
SESSION_ID	INTEGER	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.
STATEMENT_ID	VARCHAR	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session.
SESSION_TYPE	VARCHAR	Session type. Can be one of: <ul style="list-style-type: none"> • CLIENT • DBD • MERGEOUT • MOVEOUT • REBALANCE_CLUSTER • RECOVERY • REFRESH • TIMER_SERVICE • CONNECTION • SUBSESSION • REPARTITION_TABLE

Column Name	Data Type	Description
		<ul style="list-style-type: none"> LICENSE_AUDIT STARTUP SHUTDOWN VSPREAD
RUNTIME_PRIORITY	VARCHAR	Specifies how many run-time resources (CPU, I/O bandwidth) are allocated to queries that are running in the resource pool.
DESCRIPTION	VARCHAR	Transaction description in this session.
SESSION_START_TIMESTAMP	TIMESTAMPTZ	Value of session at beginning of history interval.
SESSION_END_TIMESTAMP	TIMESTAMPTZ	Value of session at end of history interval.
IS_ACTIVE	BOOLEAN	Denotes if the session is still running.
SESSION_DURATION_MS	INTEGER	Duration of the session in milliseconds.
CLIENT_TYPE	VARCHAR	Columns not used in SYSTEM_SESSIONS system table. To view values for these columns, see the V_MONITOR schema system tables SESSIONS , USER_SESSIONS , CURRENT_SESSION , and SESSION_PROFILES .
CLIENT_VERSION	VARCHAR	
CLIENT_OS	VARCHAR	
CLIENT_OS_USER_NAME	VARCHAR	The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.

Privileges

Superuser

TABLE_RECOVERIES

Provides detailed information about recovered and recovering tables during a [recovery by table](#).

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is performing the recovery.
STATUS	VARCHAR	The status of the table. Tables can have the following status: <ul style="list-style-type: none">recovered - The table is fully recoveredrecovering - The table is in the process of recoveryerror_retry - Vertica has attempted to recover the table, but the recovery failed. Tables that have not yet begun the recovery process do not have a status.
TABLE_NAME	VARCHAR	The name of the table being recovered.
TABLE_OID	INTEGER	The object ID of the table being recovered.
START_TIME	TIMESTAMPZ	The date and time that the table began recovery.
END_TIME	TIMESTAMPZ	The date and time that the table completed recovery.
PHASE	VARCHAR	The phase of the recovery .
RECOVER_PRIORITY	INTEGER	The recovery priority of the table being recovered.
THREAD_ID	VARCHAR	The ID of the thread that performed the recovery.

Privileges

None

TABLE_RECOVERY_STATUS

Provides node recovery information during a [Recovery By Table](#).

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is performing the recovery.
NODE_RECOVERY_START_TIME	TIMESTAMPZ	The timestamp for when the node began recovering.
IS_RUNNING	BOOLEAN	Indicates if the recovery process is still running.
RECOVER_EPOCH	INTEGER	The epoch that the recovery operation is trying to recover to.
RECOVERING_TABLE_NAME	VARCHAR	The name of the table currently recovering.
RECOVERED_TABLE_COUNT	INTEGER	Indicates how many tables on the node have already recovered.
TABLE_COUNT	INTEGER	The total number of tables on the node.

Privileges

None

TRANSACTIONS

Records the details of each transaction.

Column Name	Data Type	Description
START_TIMESTAMP	TIMESTAMPTZ	Beginning of history interval.
END_TIMESTAMP	TIMESTAMPTZ	End of history interval.
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.

Column Name	Data Type	Description
USER_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the user.
USER_NAME	VARCHAR	Name of the user for which transaction information is listed.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
DESCRIPTION	VARCHAR	Textual description of the transaction.
START_EPOCH	INTEGER	Number of the start epoch for the transaction.
END_EPOCH	INTEGER	Number of the end epoch for the transaction
NUMBER_OF_STATEMENTS	INTEGER	Number of query statements executed in this transaction.
ISOLATION	VARCHAR	Denotes the transaction mode as "READ COMMITTED" or "SERIALIZABLE".
IS_READ_ONLY	BOOLEAN	Denotes "READ ONLY" transaction mode.
IS_COMMITTED	BOOLEAN	Determines if the transaction was committed. False means ROLLBACK.
IS_LOCAL	BOOLEAN	Denotes transaction is local (non-distributed).
IS_INITIATOR	BOOLEAN	Denotes if the transaction occurred on this node (t).
IS_DDL	BOOLEAN	Distinguishes between a DDL transaction (t) and non-DDL transaction (f).

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

[Transactions](#)

TRUNCATED_SCHEMATA

Lists the original names of restored schemas that were truncated due to name lengths exceeding 128 characters.

Column Name	Data Type	Description
RESTORE_TIME	TIMESTAMPZ	The time that the table was restored.
SESSION_ID	VARCHAR	Identifier for the restoring session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
USER_ID	INTEGER	Identifier of the user for the restore event.
USER_NAME	VARCHAR	Name of the user for which Vertica lists restore information at the time it recorded the session.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any; otherwise NULL.
ORIGINAL_SCHEMA_NAME	VARCHAR	The original name of the schema prior to the restore.
NEW_SCHEMA_NAME	VARCHAR	The name of the schema after it was truncated.

Privileges

None

TUNING_RECOMMENDATIONS

Returns tuning recommendation results from the last [ANALYZE_WORKLOAD](#) call. This information is useful for letting you build filters on the Workload Analyzer result set.

Column Name	Data Type	Description
observatio n_count	INTEGER	Integer for the total number of events observed for this tuning recommendation. For example, if you see a return value of 1, WLA is making its first tuning recommendation for the event in 'scope'.
first_ observatio n_time	TIMESTAMPT Z	Timestamp when the event first occurred. If this column returns a null value, the tuning recommendation is from the current status of the system instead of from any prior event.
last_ observatio n_time	TIMESTAMPT Z	Timestamp when the event last occurred. If this column returns a null value, the tuning recommendation is from the current status of the system instead of from any prior event.
tuning_ parameter	VARCHAR	Objects on which you should perform a tuning action. For example, a return value of: <ul style="list-style-type: none"> public.t informs the DBA to run Database Designer on table t in the public schema bsmith notifies a DBA to set a password for user bsmith
tuning_ description	VARCHAR	Textual description of the tuning recommendation from the Workload Analyzer to perform on the tuning_ parameter object. Examples of some of the returned values include, but are not limited to: <ul style="list-style-type: none"> Run database designer on table schema.table Create replicated projection for table schema.table Consider incremental design on query Reset configuration parameter with ALTER DATABASE dbName SET parameter = value; Re-segment projection projection-name on high-cardinality column(s) Drop the projection projection-name

Column Name	Data Type	Description
		<ul style="list-style-type: none"> Alter a table's partition expression Reorganize data in partitioned table Decrease the MoveOutInterval configuration parameter setting
tuning_command	VARCHAR	<p>Command string if tuning action is a SQL command. For example, the following example statements recommend that the DBA:</p> <p>Update statistics on a particular schema's table.column: <code>SELECT ANALYZE_STATISTICS('public.table.column');</code></p> <p>Resolve mismatched configuration parameter 'LockTimeout': <code>SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'LockTimeout';</code></p> <p>Set the password for user bsmith: <code>ALTER USER (user) IDENTIFIED BY ('new_password');</code></p>
tuning_cost	VARCHAR	<p>Cost is based on the type of tuning recommendation and is one of:</p> <ul style="list-style-type: none"> LOW—minimal impact on resources from running the tuning command MEDIUM—moderate impact on resources from running the tuning command HIGH—maximum impact on resources from running the tuning command <p>Depending on the size of your database or table, consider running high-cost operations after hours instead of during peak load times.</p>

Privileges

Superuser

Examples

See [ANALYZE_WORKLOAD](#).

See Also

- [Analyzing Workloads](#)
- [Understanding WLA Triggering Conditions](#)

TUPLE_MOVER_OPERATIONS

Monitors the status of the Tuple Mover (TM) on each node. Manual mergeouts can be invoked with one of the following functions:

- [DO_TM_TASK](#)
- [PURGE](#)

Column Name	Data Type	Description
OPERATION_START_TIMESTAMP	TIMESTAMP	Start time of a Tuple Mover operation.
NODE_NAME	VARCHAR	Node name for which information is listed.
OPERATION_NAME	VARCHAR	One of the following operations: Moveout Mergeout Analyze Statistics
OPERATION_STATUS	VARCHAR	Returns Running or an empty string to indicate 'not running.' One of the following values: Start Running Complete Update Abort Change plan type to Replay Delete
TABLE_SCHEMA	VARCHAR	Schema name for the specified projection.

Column Name	Data Type	Description
TABLE_NAME	VARCHAR	Table name for the specified projection.
PROJECTION_NAME	VARCHAR	Name of the projection being processed.
PROJECTION_ID	INTEGER	Unique numeric ID assigned by the Vertica catalog, which identifies the projection.
COLUMN_ID	INTEGER	Identifier for the column for the associated projection being processed.
EARLIEST_CONTAINER_START_EPOCH	INTEGER	Populated for mergeout and purge operations only. For an ATM-invoked mergeout, for example, the returned value represents the lowest epoch of containers involved in the mergeout.
LATEST_CONTAINER_END_EPOCH	INTEGER	Populated for mergeout, and purge_partitions operations. For an ATM-invoked mergeout, for example, the returned value represents the highest epoch of containers involved in the mergeout.
ROS_COUNT	INTEGER	Number of ROS containers.
TOTAL_ROS_USED_BYTES	INTEGER	Size in bytes of all ROS containers in the mergeout operation. (Not applicable for other operations.)
PLAN_TYPE	VARCHAR	One of the following values: Moveout Mergeout Analyze Replay Delete
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	INTEGER	Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.

Column Name	Data Type	Description
IS_EXECUTING	BOOLEAN	Distinguishes between actively-running (t) and completed (f) tuple mover operations.
RUNTIME_PRIORITY	VARCHAR	Determines the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are: HIGH MEDIUM LOW

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Example

```
=> SELECT node_name, operation_status, projection_name, plan_type
      FROM TUPLE_MOVER_OPERATIONS;
node_name      | operation_status | projection_name | plan_type
-----+-----+-----+-----
v_vmart_node0001 | Running          | p1_b2          | Mergeout
v_vmart_node0002 | Running          | p1              | Mergeout
v_vmart_node0001 | Running          | p1_b2          | Replay Delete
v_vmart_node0001 | Running          | p1_b2          | Mergeout
v_vmart_node0002 | Running          | p1_b2          | Mergeout
v_vmart_node0001 | Running          | p1_b2          | Replay Delete
v_vmart_node0002 | Running          | p1              | Mergeout
v_vmart_node0003 | Running          | p1_b2          | Replay Delete
v_vmart_node0001 | Running          | p1              | Mergeout
v_vmart_node0002 | Running          | p1_b1          | Mergeout
```

See Also

- [DO_TM_TASK](#)
- [PURGE](#)

UDX_FENCED_PROCESSES

Provides information about processes Vertica uses to run user-defined extensions in fenced mode.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
PROCESS_TYPE	VARCHAR	Indicates what kind of side process this row is for and can be one of the following values: <ul style="list-style-type: none"> UDxZygoteProcess — Master process that creates worker side processes, as needed, for queries. There will be, at most, 1 UP UDxZygoteProcess for each Vertica instance. UDxSideProcess — Indicates that the process is a worker side process. There could be many UDxSideProcesses, depending on how many sessions there are, how many queries, and so on.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
LANGUAGE	VARCHAR	The language of the UDX. For example 'R' or 'C++';
MAX_MEMORY_JAVA_KB	INT	The maximum amount of memory in KB that can be used for the Java heap file on the node.
PID	INTEGER	Linux process identifier of the side process (UDxSideProcess).
PORT	VARCHAR	For Vertica internal use. The TCP port that the side process is listening on.
STATUS	VARCHAR	Set to UP or DOWN, depending on whether the process is alive or not. After a process fails, Vertica restarts it only on demand.

Column Name	Data Type	Description
		So after a process failure, there might be periods of time when no side processes run.

Privileges

None

USER_LIBRARIES

Lists the user libraries that are currently loaded. These libraries contain user-defined extensions (UDxs) that provide additional analytic functions.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR (8192)	The name of the schema containing the library.
LIB_NAME	VARCHAR (8192)	The name of the library.
LIB_OID	INTEGER	The object ID of the library.
AUTHOR	VARCHAR (8192)	The creator of the library file.
OWNER_ID	INTEGER	The object ID of the library's owner.
LIB_FILE_NAME	VARCHAR (8192)	The name of the shared library file.
MD5_SUM	VARCHAR (8192)	The MD5 checksum of the library file, used to verify that the file was correctly copied to each node. Note: This use of MD5 is not for cryptographic or authentication purposes. For information on authenticating with MD5 see Hash Authentication .
SDK_VERSION	VARCHAR (8192)	The version of the Vertica SDK used to compile the library.

Column Name	Data Type	Description
REVISION	VARCHAR (8192)	The revision of the Vertica SDK used to compile the library.
LIB_BUILD_TAG	VARCHAR (8192)	Internal information set by library developer to track the when the library was compiled.
LIB_VERSION	VARCHAR (8192)	The version of the library.
LIB_SDK_VERSION	VARCHAR (8192)	The version of the Vertica SDK intended for use with the library. The developer sets this value manually. This value may differ from the values in the SDK_VERSION and REVISION, which are set automatically during compilation.
SOURCE_URL	VARCHAR (8192)	A URL that contains information about the library.
DESCRIPTION	VARCHAR (8192)	A description of the library.
LICENSES_REQUIRED	VARCHAR (8192)	The licenses required to use the library.
SIGNATURE	VARCHAR (8192)	The signature used to sign the library for validation.
DEPENDENCIES	VARCHAR (8192)	External libraries on which this library depends. These libraries are maintained by Vertica, just like the user libraries themselves.

USER_LIBRARY_MANIFEST

Lists user-defined functions contained in all loaded user libraries.

Column Name	Data Type	Description
SCHEMA_NAME	VARCHAR	The name of the schema containing the function.

Column Name	Data Type	Description
LIB_NAME	VARCHAR	The name of the library containing the UDF.
LIB_OID	INTEGER	The object ID of the library containing the function.
OBJ_NAME	VARCHAR	The name of the constructor class in the library for a function.
OBJ_TYPE	VARCHAR	The type of user defined function (scalar function, transform function)
ARG_TYPES	VARCHAR	A comma-delimited list of data types of the function's parameters.
RETURN_TYPE	VARCHAR	A comma-delimited list of data types of the function's return values.

Privileges

None

USER_SESSIONS

Returns user session history on the system.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	Name of the node that is reporting the requested information.
USER_NAME	VARCHAR	Name of the user at the time Vertica recorded the session.
SESSION_ID	VARCHAR	Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
TRANSACTION_ID	VARCHAR	Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.

Column Name	Data Type	Description
STATEMENT_ID	VARCHAR	Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session.
RUNTIME_PRIORITY	VARCHAR	<p>Determines the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to queries already running in the resource pool. Valid values are:</p> <ul style="list-style-type: none"> • HIGH • MEDIUM • LOW <p>Queries with a HIGH run-time priority are given more CPU and I/O resources than those with a MEDIUM or LOW run-time priority.</p>
SESSION_START_TIMESTAMP	TIMESTAMPTZ	Value of session at beginning of history interval.
SESSION_END_TIMESTAMP	TIMESTAMPTZ	Value of session at end of history interval.
IS_ACTIVE	BOOLEAN	Denotes if the operation is executing.
CLIENT_HOSTNAME	VARCHAR	IP address of the client system
CLIENT_PID	INTEGER	<p>Linux process identifier of the client process that issued this connection.</p> <p>Note: The client process could be on a different machine from the server.</p>
CLIENT_LABEL	VARCHAR	User-specified label for the client connection that can be set when using ODBC. See Label in DSN Parameters in Connecting to Vertica.
SSL_STATE	VARCHAR	<p>Indicates if Vertica used Secure Socket Layer (SSL) for a particular session. Possible values are:</p> <ul style="list-style-type: none"> • None – Vertica did not use SSL.

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • Server – Server authentication was used, so the client could authenticate the server. • Mutual – Both the server and the client authenticated one another through mutual authentication. <p>See Implementing Security and TLS/SSL Server Authentication in the Administrator's Guide.</p>
AUTHENTICATION_METHOD	VARCHAR	<p>Type of client authentication used for a particular session, if known. Possible values are:</p> <ul style="list-style-type: none"> • Unknown • Trust • Reject • Kerberos • Password • MD5 • LDAP • Kerberos-GSS • Ident <p>See Security and Authentication and Implementing Client Authentication.</p>
CLIENT_TYPE	VARCHAR	<p>The type of client from which the connection was made. Possible client type values:</p> <ul style="list-style-type: none"> • ADO.NET Driver • ODBC Driver • JDBC Driver • vsql

Column Name	Data Type	Description
CLIENT_VERSION	VARCHAR	Returns the client version.
CLIENT_OS	VARCHAR	Returns the client operating system.
CLIENT_OS_USER_NAME	VARCHAR	The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

See Also

- [CURRENT_SESSION](#)
- [SESSION_PROFILES](#)
- [SESSIONS](#)
- [SYSTEM_SESSIONS](#)

WOS_CONTAINER_STORAGE

Monitors information about WOS storage, which is divided into regions. Each region allocates blocks of a specific size to store rows.

Note: The WOS allocator can use large amounts of virtual memory without assigning physical memory.

Column Name	Data Type	Description
NODE_NAME	VARCHAR	The node name for which information is listed.
WOS_TYPE	VARCHAR	Returns one of the following:

Column Name	Data Type	Description
		<ul style="list-style-type: none"> • <code>system</code> – for system table queries • <code>user</code> – for other user queries
<code>WOS_ALLOCATION_REGION</code>	<code>VARCHAR</code>	The block size allocated by region in KB. The summary line sums the amount of memory used by all regions.
<code>REGION_VIRTUAL_SIZE_KB</code>	<code>INTEGER</code>	The amount of virtual memory in use by region in KB. Virtual size is greater than or equal to allocated size, which is greater than or equal to in-use size.
<code>REGION_ALLOCATED_SIZE_KB</code>	<code>INTEGER</code>	The amount of physical memory in use by a particular region in KB.
<code>REGION_IN_USE_SIZE_KB</code>	<code>INTEGER</code>	The actual amount of data stored by the region in KB. The amount of memory used by the WOS is typically capped at one quarter physical memory per node.
<code>REGION_SMALL_RELEASE_COUNT</code>	<code>INTEGER</code>	Internal use only
<code>REGION_BIG_RELEASE_COUNT</code>	<code>INTEGER</code>	Internal use only
<code>EXTRA_RESERVED_BYTES</code>	<code>INTEGER</code>	The amount of extra memory allocated to maintain WOS sort information.
<code>EXTRA_USED_BYTES</code>	<code>INTEGER</code>	The amount of memory in use currently to maintain the WOS sort information.

Appendix: Compatibility with Other RDBMS

This section describes compatibility of Vertica with other relational database management systems.

Information in this appendix is intended to simplify database migration to Vertica.

Data Type Mappings Between Vertica and Oracle

Oracle uses proprietary data types for all main data types (for example, VARCHAR, INTEGER, FLOAT, DATE), if you plan to migrate your database from Oracle to Vertica, Micro Focus strongly recommends that you convert the schema—a simple and important exercise that can minimize errors and time lost spent fixing erroneous data issues.

The following table compares the behavior of Oracle data types to Vertica data types.

Oracle	Vertica	Notes
NUMBER (no explicit precision)	INT, NUMERIC or FLOAT	<p>In Oracle, the NUMBER data type with no explicit precision stores each number N as an integer M, together with a scale S. The scale can range from -84 to 127, while the precision of M is limited to 38 digits. So $N = M * 10^S$.</p> <p>When precision is specified, precision/scale applies to all entries in the column. If omitted, the scale defaults to 0.</p> <p>For the common case where Oracle's NUMBER with no explicit precision data type is used to store only integer values, INT is the best suited and the fastest Vertica data type. However, INT (the same as BIGINT) is limited to a little less than 19 digits, with a scale of 0; if the Oracle column contains integer values outside of the range [-9223372036854775807, +9223372036854775807], use the Vertica data type NUMERIC(p,0) where p is the maximum number of digits required to represent the values of N.</p> <p>Even though no explicit scale is specified for an Oracle</p>

Oracle	Vertica	Notes
		<p>NUMBER column, Oracle allows non-integer values, each with its own scale. If the data stored in the column is approximate, Vertica recommends using the Vertica data type FLOAT, which is standard IEEE floating point, like ORACLE BINARY_DOUBLE. If the data is exact with fractional places, for example dollar amounts, Vertica recommends NUMERIC(p,s) where p is the precision (total number of digits) and s is the maximum scale (number of decimal places).</p> <p>Vertica conforms to standard SQL, which requires that $p \geq s$ and $s \geq 0$. Vertica's NUMERIC data type is most effective for $p=18$, and increasingly expensive for $p=37, 58, 67$, etc., where $p \leq 1024$.</p> <p>Vertica recommends against using the data type NUMERIC(38,s) as a default "failsafe" mapping to guarantee no loss of precision. NUMERIC(18,s) is better, and INT or FLOAT are better yet, if one of these data types will do the job.</p>
NUMBER (P,0), P ≤ 18	INT	In Oracle, when precision is specified the precision/scale applies to all entries in the column. If omitted the scale defaults to 0. For the Oracle NUMBER data type with 0 scale, and a precision less than or equal to 18, use INT in Vertica.
NUMBER (P,0), P > 18	NUMERIC (p,0)	<p>An Oracle column precision greater than 18 is often more than an application really needs.</p> <p>If all values in the Oracle column are within the INT range [-9223372036854775807,+9223372036854775807], use INT for best performance. Otherwise, use the Vertica data type NUMERIC(p, 0), where $p = P$.</p>
NUMBER (P,S) all cases other than previous 3 rows	NUMERIC (p,s) or FLOAT	<p>When $P \geq S$ and $S \geq 0$, use $p = P$ and $s = S$, unless the data allows reducing P or using FLOAT as discussed above.</p> <p>If $S > P$, use $p = S$, $s = S$. If $S < 0$, use $p = P - S$, $s = 0$.</p>
NUMERIC	See notes -->	Rarely used in Oracle. See notes for the NUMBER type.

Oracle	Vertica	Notes
(P,S)		
DECIMAL (P,S)	See notes -->	DECIMAL is a synonym for NUMERIC. See notes for the NUMBER type.
BINARY_ FLOAT	FLOAT	Same as FLOAT(53) or DOUBLE PRECISION.
BINARY_ DOUBLE	FLOAT	Same as FLOAT(53) or DOUBLE PRECISION.
RAW	VARBINARY (RAW)	The maximum size of RAW in Oracle is 2,000 bytes. The maximum size of CHAR/BINARY in Vertica is 65000 bytes. In Vertica, RAW is a synonym for VARBINARY.
LONG RAW	VARBINARY (RAW)	The maximum size of Oracle's LONG RAW is 2GB. The maximum size of Vertica's VARBINARY is 65000 bytes. Vertica users should exercise caution to avoid truncation during data migration from Oracle.
CHAR(n)	CHAR(n)	The maximum size of CHAR in Oracle is 2,000 bytes. The maximum size of CHAR in Vertica is 65000 bytes.
NCHAR(n)	CHAR(n*3)	Vertica supports national characters with CHAR(n) as variable-length UTF8-encoded UNICODE character string. UTF-8 represents ASCII in 1 byte, most European characters in 2 bytes, and most oriental and Middle Eastern characters in 3 bytes.
VARCHAR2 (n)	VARCHAR(n)	The maximum size of VARCHAR2 in Oracle is 4,000 bytes. The maximum size of VARCHAR in Vertica is 65000. Note: The behavior of Oracle's VARCHAR2 and Vertica's VARCHAR is semantically different. Vertica's VARCHAR exhibits standard SQL behavior, whereas Oracle's VARCHAR2 is not completely consistent with standard behavior – it treats an empty string as NULL value and uses non-padded comparison if one operand is VARCHAR2.
NVARCHAR2	VARCHAR	See notes for NCHAR().

Oracle	Vertica	Notes
(n)	(n*3)	
DATE	TIMESTAMP or possibly DATE	Oracle's DATE is different from the SQL standard DATE data type implemented by Vertica. Oracle's DATE includes the time (no fractional seconds), while Vertica DATE type includes only date per SQL specification.
TIMESTAMP	TIMESTAMP	TIMESTAMP defaults to six places, that is, to microseconds
TIMESTAMP WITH TIME_ZONE	TIMESTAMP WITH TIME_ZONE	TIME_ZONE defaults to the currently SET or system time zone.
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	Per the SQL standard, INTERVAL can be qualified with YEAR TO MONTH sub-type in Vertica.
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	In Vertica, DAY TO SECOND is the default sub-type for INTERVAL.
CLOB, BLOB	LONG VARCHAR, LONG VARBINARY	You can store a CLOB (character large object) or BLOB (binary large object) value in a table or in an external location. The maximum size of a CLOB or BLOB is 128 TB. You can store Vertica LONG data types only in LONG VARCHAR and LONG VARBINARY columns. The maximum size of the LONG data types is 32,000,000 bytes.
LONG, LONG RAW	LONG VARCHAR, LONG VARBINARY	Oracle recommends using CLOB and BLOB data types instead of LONG and LONG RAW data types. In Oracle, a table can contain only one LONG column, The maximum size of a LONG or LONG RAW data type is 2 GB.

Security and Authentication

Vertica provides tools and features that allow you to ensure your system is secure as well as to prevent unauthorized users from accessing sensitive information.

[Client Authentication](#) establish the identity of the requesting client and determines whether that client is authorized to connect to the Vertica server.

Client Authentication

Implementing strong security programs provides Vertica users the assurance that access to sensitive information is closely guarded. Vertica uses several approaches to manage data access.

The database server uses client authentication to establish the identity of the requesting client and determines whether that client is authorized to connect to the Vertica server using the supplied credentials.

Encrypting Client-Server Communication

Vertica uses Secure Socket Layer (SSL) and [Transport Layer Security \(TLS\)](#) to establish a secure connection between the client machine and the server. Configure SSL/TLS to:

- Authenticate the server so the client can confirm the server's identity. Vertica supports mutual authentication in which the server can also confirm the identity of the client. This authentication helps prevent "man-in-the-middle" attacks.
- Encrypt data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised.
- Verify that data sent between the client and server has not been altered during transmission.

For details see [TLS/SSL Server Authentication](#).

Authentication Management

Users with the [DBADMIN Role](#) can manage the following authentication tasks:

- Create authentication records using [CREATE AUTHENTICATION](#).

Important: Configure client authentication so that the DBADMIN user can always access the database locally. If a problem occurs with the authentication that blocks all users from logging in, the DBADMIN user needs access to correct the problem.

- Assign a specific authentication method to a user using [GRANT \(Authentication\)](#).
- Use [ALTER AUTHENTICATION](#) to:
 - Enable/disable authentication methods.
 - Define a default authentication method to be used if a user has not been assigned a specific authentication method.
- Define parameters required by [LDAP](#), [Ident](#), and [Kerberos](#) authentication methods.
- Revoke a user's authentication record using [REVOKE Authentication](#). This user now uses the default authentication.
- Delete an authentication record from the database using [DROP AUTHENTICATION](#). Any users assigned the dropped record now use the default authentication method.

For details about managing authentication records, see:

- [DBADMIN Authentication Access](#)
- [Creating Authentication Records](#)
- [Enabling and Disabling Authentication Methods](#)
- [Granting and Revoking Authentication Methods](#)
- [Modifying Authentication Records](#)

See [Implementing Client Authentication](#).

User Authorization

Database users should have access to just the database resources they need to perform their required tasks. For example, some users need to query only specific sets of data. To prevent unauthorized access to additional data, you can limit their access to just the data that they need to run their queries. Other users should be able to read the data but not be able to modify or insert new data. Still other users might need more permissive access, including the

right to create and modify schemas, tables, and views, or grant other users access to database resources.

For information on controlling data access, see the following:

- [About Database Users](#) in [Managing Users and Privileges](#)
- [About Database Roles](#) to grant users access to a set of privileges.
- [Access Policies](#) to limit user's from viewing data from a specific table.

Implementing Client Authentication

Vertica restricts which database users can connect through **client authentication**. The database server uses client authentication to establish the identity of the requesting client and determines whether that client is authorized to connect to the Vertica server using the supplied credentials.

When a user or client application connect to the Vertica database server, it supplies a unique user name and password to gain access.

Vertica offers several client authentication methods. You can configure Vertica to require just a user name for connections, you likely require more secure means of authentication, such as a password at a minimum.

Note: Topics in this section describe authentication methods supported at the database server layer. For information on authentication between server and client, see [TLS/SSL Server Authentication](#).

How Client Authentication Works

When connecting to a Vertica database, a user or client application must supply the name of a valid user account. In addition, the application usually includes a means of authentication, such as a password or security certificate.

There are two types of client authentication:

- **LOCAL**—Authenticating users or applications that are trying to connect from the same node that the database is running on.

- HOST—Authenticating users or applications that are trying to connect from a node that has a different IPv4 or IPv6 address than the database.

The DBADMIN user manages the client authentication information that the database uses to authenticate users.

Vertica takes the following steps to authenticate users:

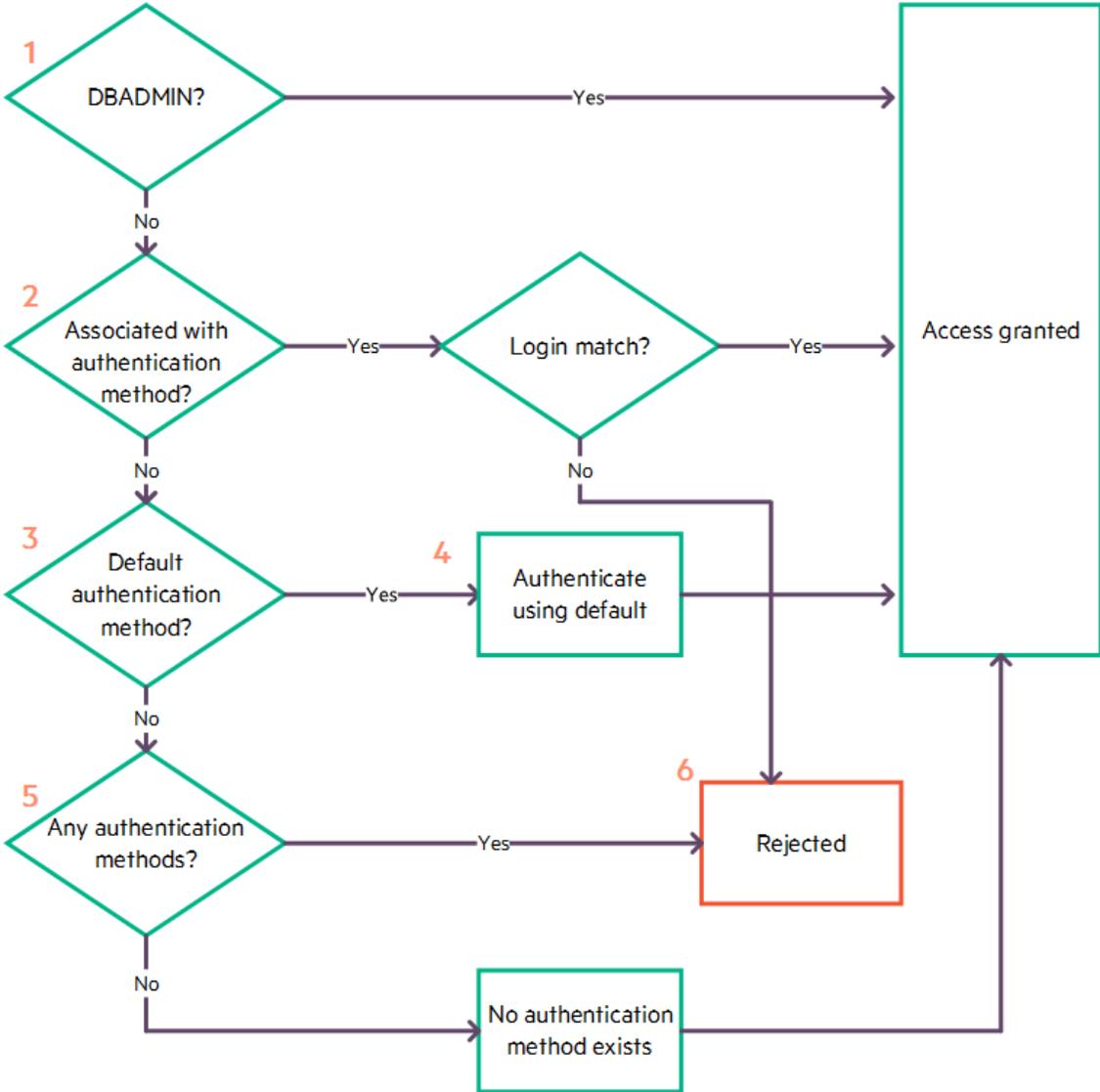
1. When a user or application attempts to connect to a Vertica database, the system checks to see if the user is a DBADMIN user. If so, authentication occurs using the assigned authentication method, local trust or local hash authentication.
2. For non-DBADMIN users, the database checks to see if the user is associated with an authentication method through a GRANT statement. If so, the database allows the user to log in if they match the parameters required for that authentication method.

Note: For detailed information on how authentication priorities work, see [Priorities for Client Authentication Methods](#).

The DBADMIN user can grant an authentication method to users or user roles. The DBADMIN user can also create a default authentication method that Vertica uses when no authentication has been associated with a user or role.

3. If the user has not been granted an authentication method, the database checks to see if the DBADMIN has established a default authentication method.
4. If the DBADMIN has specified a default authentication method, the database authenticates the user using that default method.
5. If you have not specified a default authentication method, the database checks to see if the DBADMIN user has defined any authentication methods. If not, no authentication information exists in the database. However, if a password exists, the DBADMIN user can log in.
6. If authentication information exists, Vertica rejects the user request to connect to the database. The DBADMIN has not granted an authentication method for that user nor has the DBADMIN defined a default authentication method for all users ('public ').
7. If no authentication records exist in the database, Vertica uses implicit trust/implicit password to authenticate the user.

The following image illustrates the steps involved in client authentication:



IPv4 and IPv6 for Client Authentication

Vertica supports clients using either the IPv4 or the IPv6 protocol to connect to the database server. Internal communication between database servers must consistently use one address family (IPv4 or IPv6). The client, however, can connect to the database from either type of IP address.

If the client will be connecting from either IPv4 or IPv6, you must create two authentication methods, one for each address. Any authentication method that uses HOST authentication requires an IP address.

For example, the first statement allows users to connect from any IPv4 address. The second statement allows users to connect from any IPv6 address:

```
=> CREATE AUTHENTICATION <name> METHOD 'gss' HOST '0.0.0.0/0; --IPv4  
=> CREATE AUTHENTICATION <name> METHOD 'gss' HOST '::/0; --IPv6
```

If you are using a literal IPv6 address in a URL, you must enclose the IPv6 address in square brackets as shown in the following examples:

```
=> ALTER AUTHENTICATION ldap SET host='ldap://[1dfa:2bfa:3:45:5:6:7:877]';  
=> ALTER AUTHENTICATION ldap SET host='ldap://[fdb:dbfa:0:65::177]';  
=> ALTER AUTHENTICATION ldap SET host='ldap://[fdb::177]';  
=> ALTER AUTHENTICATION ldap SET host='ldap://[::1]';  
=> ALTER AUTHENTICATION ldap SET host='ldap://[1dfa:2bfa:3:45:5:6:7:877]:5678';
```

If you are working with a multi-node cluster, any IP/netmask settings in (HOST, HOST TLS, HOST NO TLS) must match all nodes in the cluster. This setup allows the database owner to authenticate with and administer every node in the cluster. For example, specifying 10.10.0.8/30 allows a CIDR address range of 10.10.0.8–10.10.0.11.

For detailed information about IPv6 addresses, see [RFC 1924](#) and [RFC 2732](#).

Supported Client Authentication Methods

Vertica supports the following types of authentication to prove a client's identity.

- Trust—Authorizes any user that connects to the database using a valid user name, no password is required and authentication is not performed.
- Reject—Rejects the connection attempt when a user with an invalid user name attempts to connect to the database.
- Kerberos (GSS)—Authorizes connecting to the database using a MIT Kerberos implementation. The KDC must support Kerberos 5 using GSS-API. This API also provides compatibility with non-MIT Kerberos implementations, such as Java and Windows clients.
- Hash—Sends encrypted passwords hashed by the MD5 algorithm or the more secure SHA-512 method over the network. The server provides the client with salt.
- LDAP—Works like password authentication except the LDAP method authenticates the client against a Lightweight Directory Access Protocol or Active Directory server.
- Ident—Authenticates the client against the username in an Ident server.

- TLS authentication—Authenticates the client using digital certificates that contain a public key. Transport Layer Security (TLS) is the successor to Secure Sockets Layer (SSL) authentication.

Local and Host Authentication

You can define a client authentication method as:

- Local: Local connection to the database.
- Host: Remote connection to the database from different hosts, each with their own IPv4 or IPv6 address and host parameters. For more information see [IPv4 and IPv6 for Client Authentication](#) above.

Some authentication methods cannot be designated as local, as listed in this table:

Authentication Method	Local?	Host?
Kerberos (GSS)	No	Yes
Ident	Yes	No
LDAP	Yes	Yes
Hash	Yes	Yes
Reject	Yes	Yes
Trust	Yes	Yes

Authentication for Chained Users and Roles

Vertica supports creating chained users and roles, where you can grant ROLE2 privileges to ROLE1. All users in ROLE1 use the same authentication assigned to ROLE2. For example:

```
=> CREATE USER user1;
=> CREATE ROLE role1;
=> CREATE ROLE role2;
=> CREATE AUTHENTICATION h1 method 'hash' local;
=> GRANT AUTHENTICATION h1 to role2;
=> GRANT role2 to role1;
=> GRANT role1 to user1;
```

The user and role chain in the example above can be illustrated as follows:

auth1 -> role2 -> role1 -> user1

In this example, since role2 privileges are granted to role1 you only need to grant authentication to role2 to also enable it for role1.

DBADMIN Authentication Access

The DBADMIN user must have access to the database at all times.

The DBADMIN account must authenticate against the database using local trust or local hash authentication.

Micro Focus recommends that you create an authentication method (LOCAL TRUST or LOCAL PASSWORD) with a very high priority, say, 10,000. Grant this method to the DBADMIN user and set the priority using ALTER AUTHENTICATION.

With the high priority, this new authentication method supersedes any authentication methods you create for PUBLIC (which includes the DBADMIN user). Even if you make changes to PUBLIC authentication methods, the DBADMIN user can now connect to the database at any time.

This example shows how you configure local trust authentication for the DBADMIN user. As a result, the user can use `vsq1` with the `-h` option and does not need to enter a password:

```
=> CREATE AUTHENTICATION v_dbadmin_trust METHOD 'trust' LOCAL;  
=> GRANT AUTHENTICATION v_dbadmin_trust TO dbadmin;  
=> ALTER AUTHENTICATION v_dbadmin_trust PRIORITY 10000;
```

The next example shows how you configure host hash authentication for DBADMIN. They allow the user to access the Vertica database using the assigned password from any IPv4 address.

The DBADMIN user can access the database using `vsq1 -h Hostname --host Hostname`, the Administration Tools, or any other tools that connects to Vertica:

```
=> CREATE AUTHENTICATION v_dbadmin_hash METHOD 'hash' HOST '0.0.0.0/0';  
=> GRANT AUTHENTICATION v_dbadmin_hash TO dbadmin;  
=> ALTER AUTHENTICATION v_dbadmin_hash PRIORITY 10000;  
=> SELECT SET_CONFIG_PARAMETER('SecurityAlgorithm', 'SHA512');
```

Note: Vertica supports IPv4 and IPv6 addresses. For more information, see [IPv4 and IPv6 for Client Authentication](#).

Creating Authentication Records

You can manage client authentication records using `vsq` commands. To use these statements, you must be connected to the database.

Important: You cannot modify client authentication records using the Administration Tools. The Administration Tools interface allows you to modify the contents of the `vertica.conf` file. However, Vertica ignores any client authentication information stored in that file.

When you create authentication records using [CREATE AUTHENTICATION](#), specify the following information.

What you need to specify	Description
Authentication method name	A name that you define for Vertica use.
Authentication type	The type of authentication Vertica should use to validate the user or client attempting to connect: <ul style="list-style-type: none">• 'gss'• 'ident'• 'ldap'• 'hash'• 'reject'• 'trust'• 'tls'
Access method	<ul style="list-style-type: none">• LOCAL• HOST• HOST NO TLS• HOST TLS

What you need to specify	Description
Host IP address	IP address or range of IP addresses from which the user or application tries to connect. This can be an IPv4 address or an IPv6 address. For more information, see IPv4 and IPv6 for Client Authentication .

The following examples show how to create authentication records that are stored in the catalog. When you create an authentication record using CREATE AUTHENTICATION, Vertica automatically enables it.

This example shows you how to create an authentication method named `localpwd` to authenticate users who are trying to log in from a local host using a password:

```
=> CREATE AUTHENTICATION localpwd METHOD 'hash' LOCAL;
```

This example shows you how to create an authentication method named `v_ldap` that uses LDAP over TLS to authenticate users logging in from the host with the IPv4 address `10.0.0.0/23`:

```
=> CREATE AUTHENTICATION v_ldap METHOD 'ldap' HOST TLS '10.0.0.0/23';
```

This example shows you how to create an authentication method named `v_kerberos` to authenticate users that are trying to connect from any host in the networks `2001:0db8:0001:12xx`:

```
=> CREATE AUTHENTICATION v_kerberos METHOD 'gss' HOST '2001:db8:1::1200/56';
```

This example shows you how to create an authentication method named, `RejectNoSSL`, that rejects users from any IP address that are trying to authenticate without SSL/TLS:

```
=> CREATE AUTHENTICATION RejectNoSSL_IPv4 METHOD 'reject' HOST NO TLS '0.0.0.0/0'; --IPv4  
=> CREATE AUTHENTICATION RejectNoSSL_IPv6 METHOD 'reject' HOST NO TLS '::/0'; --IPv6
```

See Also

- [Deleting Authentication Records](#)
- [Enabling and Disabling Authentication Methods](#)
- [Granting and Revoking Authentication Methods](#)
- [Modifying Authentication Records](#)

Modifying Authentication Records

To modify existing authentication records, you must first be connected to your database. The following examples show how to make changes to your authentication records. For more information see [ALTER AUTHENTICATION](#).

Rename an Authentication Method

Rename the `v_kerberos` authentication method to `K5`, and enable it. All users who have been associated with the `v_kerberos` authentication method are now associated with the `K5` method granted instead.

```
=> ALTER AUTHENTICATION v_kerberos RENAME TO K5 ENABLE;
```

Specify a Priority for an Authentication Method

Specify a priority of 10 for `K5` authentication:

```
=> ALTER AUTHENTICATION K5 PRIORITY 10;
```

For more information see [Priorities for Client Authentication Methods](#).

Change a Parameter

Set the `system_users` parameter for `ident1` authentication to `root`:

```
=> CREATE AUTHENTICATION ident1 METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION ident1 SET system_users='root';
```

Change the IP address and specify the parameters for an LDAP authentication method named `Ldap1`.

In this example, you specify the bind parameters for the LDAP server. Vertica connects to the LDAP server, which authenticates the Vertica client. If the authentication succeeds, Vertica authenticates any users who have been granted the Ldap1 authentication method on the designated LDAP server:

```
=> CREATE AUTHENTICATION Ldap1 METHOD 'ldap' HOST '172.16.65.196';  
=> ALTER AUTHENTICATION Ldap1 SET host='ldap://172.16.65.177',  
    binddn_prefix='cn=', binddn_suffix=',dc=qa_domain,dc=com';
```

Change the IP address, and specify the parameters for an LDAP authentication method named Ldap1. Assume that Vertica does not have enough information to create the distinguished name (DN) for a user attempting to authenticate. Therefore, in this case, you must specify to use LDAP search and bind:

```
=> CREATE AUTHENTICATION LDAP1 METHOD 'ldap' HOST '172.16.65.196';  
=> ALTER AUTHENTICATION Ldap1 SET host='ldap://172.16.65.177',  
    basedn='dc=qa_domain,dc=com', binddn='cn=Manager,dc=qa_domain,  
    dc=com', search_attribute='cn', bind_password='secret';
```

Change the Associated Method

Change the localpwd authentication from trust to hash:

```
=> CREATE AUTHENTICATION localpwd METHOD 'trust' LOCAL;  
=> ALTER AUTHENTICATION localpwd METHOD 'hash';
```

ALTER AUTHENTICATION validates the parameters you enter. If there are errors, it disables the authentication method that you are trying to modify.

Using the Administration Tools

The advantages of using the Administration Tools are:

- You do not have to connect to the database
- The editor verifies that records are correctly formed
- The editor maintains records so they are available to you to edit later

Note: You must restart the database to implement your changes.

For information about using the Administration Tools to create and edit authentication records, see [Creating Authentication Records](#).

Using the Client Authentication Configuration Parameter

The advantage of using the `ClientAuthentication` configuration parameter is that the changes are implemented immediately across all nodes within the database cluster. You do not need to restart the database.

However, all the database nodes must be up and you must [connect to the database](#) before you set this parameter. Most importantly, this method does not verify that records are correctly formed and it does not maintain the records so you can modify them later.

New authentication records are appended to the list of existing authentication records. Because Vertica scans the list of records from top to bottom and uses the first record that matches the incoming connection, you might find your newly-added record does not have an effect if Vertica used an earlier record instead.

To configure client authentication through a connection parameter, use the `ALTER DATABASE` statement:

```
=> ALTER DATABASE exampleddb SET ClientAuthentication = 'connection type user name address method';
```

When you specify authentication records, make sure to adhere to the following guidelines:

- Fields that make up the record can be separated by white space or tabs
- Other than IP addresses and mask columns, field values cannot contain white space

Examples

The following example creates an authentication record for the trust method:

```
=> ALTER DATABASE exampleddb SET ClientAuthentication = 'hostnossl dbadmin 0.0.0.0/0 trust';
```

The following example creates an authentication record for the LDAP method:

```
=> ALTER DATABASE exampleddb SET ClientAuthentication = 'host all 10.0.0.0/8  
ldap "ldap://summit.vertica.com;cn=;,dc=vertica,dc=com";
```

The following example specifies three authentication records. In a single command, separate each authentication record by a comma:

```
=> ALTER DATABASE exampledb SET ClientAuthentication = 'hostnossl dbadmin 0.0.0.0/0 trust, hostnossl  
all 0.0.0.0/0 md5,  
local all trust';
```

Deleting Authentication Records

To delete client authentication record, use [DROP AUTHENTICATION](#). To use this approach, you have to be connected to your database.

To delete an authentication record for md5_auth use the following command:

```
=> DROP AUTHENTICATION md5_auth;
```

To delete an authentication record for a method that has been granted to a user, use the **CASCADE** keyword:

```
=> CREATE AUTHENTICATION localpwd METHOD 'password' LOCAL;  
=> GRANT AUTHENTICATION localpwd TO jsmith;  
=> DROP AUTHENTICATION localpwd CASCADE;
```

See Also

- [Creating Authentication Records](#)
- [Granting and Revoking Authentication Methods](#)

Priorities for Client Authentication Methods

You can associate one or more authentication methods to a connection or user. For a user who has multiple authentication methods, specify the order in which Vertica should try them. To do so, assign a priority to each authentication method using [ALTER AUTHENTICATION](#). All priority values should be a non-negative INTEGER.

Higher values indicate higher priorities. Vertica tries to authenticate a user with an authentication method in order of priority from highest to lowest. For example:

- A priority of 10 is higher than a priority of 5.
- A priority 0 is the lowest possible value.

Priority Order for Authentication Methods

When you associate multiple authentication methods with a connection, Vertica uses the following order to determine how to authenticate the client:

- Administrator-assigned priority for an individual method
- The most specific IP addresses have priority over the least specific IP addresses

For example, the IPv4 address 10.3.4.128/25 has priority over 10.3.0.0/24, which in turn has priority over 10.3.0.0/16. The IPv6 address 2001:db8:ab::123/128 has priority over 2001:db8:1::1200/56.

- Reject
- GSS | LDAP | Ident
- Hash
- Trust

Authentication Attempts Using Multiple Methods

If there is only one authentication method associated with a user, Vertica uses that method to authenticate the login attempt.

If the administrator has associated multiple authentication methods with a given user or IP address, Vertica tries to authenticate as follows:

- If the highest priority authentication method is Ident and authentication fails, Vertica tries the next highest priority authentication method, regardless of what method it uses.

If the next attempt does not use Ident authentication and fails, the authentication process ends. However, if the next attempt uses Ident and fails, Vertica continues to the next highest priority method. This process continues until authentication is successful or a non-Ident authentication attempt fails.

- If the highest priority method is LDAP and authentication fails, Vertica searches for the next highest priority LDAP method. Authentication attempts continue until the authentication is successful, or there are no additional LDAP authentication methods that satisfy the connection criteria.

Note that if a user not found error occurs during LDAP authentication, the retry connection attempt initiates only if you set the `ldap_continue` parameter to yes.

- For all other authentication types, Vertica tries the highest priority authentication method associated with that user. If that authentication fails, the authentication process stops.

For example, suppose there are two client authentication methods associated with a user, as follows:

```
=> CREATE AUTHENTICATION auth_name1 METHOD 'hash' LOCAL;  
=> GRANT AUTHENTICATION auth_name1 to user;  
=> ALTER AUTHENTICATION auth_name1 PRIORITY 5;  
=> CREATE AUTHENTICATION auth_name2 METHOD 'ident' LOCAL;  
=> GRANT AUTHENTICATION auth_name2 to user;  
=> ALTER AUTHENTICATION auth_name2 PRIORITY 10;
```

When user tries to connect to the database, Vertica first tries `auth_name2` to authenticate because it has a higher priority. If that fails, Vertica tries `auth_name1`. If that fails, authentication fails.

Specifying Authentication Method Priority

To specify priorities for client authentication methods, use [ALTER AUTHENTICATION](#). The priority value must be a non-negative INTEGER. Higher numbers indicate a higher priority. The default value, 0, is the lowest possible priority.

The syntax is:

```
ALTER AUTHENTICATION <name> ... PRIORITY <priority_value>;
```

If you do not specify a priority, or omit the `<priority_value>` when using `ALTER AUTHENTICATION`, Vertica sets the priority to 0.

DBADMIN and Authentication Priority

To allow the DBADMIN user to connect to the database at any time, Micro Focus recommends that you create an authentication method (LOCAL TRUST or LOCAL PASSWORD) with a very high priority, such as 10,000. Grant this method to the DBADMIN user, and set the priority using `ALTER AUTHENTICATION`.

With the high priority, this new authentication method supersedes any authentication methods you create for PUBLIC (which includes the DBADMIN user). Even if you make changes to PUBLIC authentication methods, the DBADMIN still has access.

Note: For the DBADMIN user to be able to perform all Admintools functions, the DBADMIN must always be able to authenticate by LOCAL TRUST or LOCAL PASSWORD (the default for DBADMIN user). If you have changed DBADMIN user authentication from LOCAL TRUST or LOCAL PASSWORD, use the [ALTER AUTHENTICATION](#) statement to once again give the DBADMIN user LOCAL TRUST or LOCAL PASSWORD authentication.

Viewing Information About Client Authentication Records

For information about client authentication records that you have configured for your database, query the following system tables in the V_CATALOG schema:

- [CLIENT_AUTH](#)
- [CLIENT_AUTH_PARAMS](#)
- [PASSWORD_AUDITOR](#)
- [USER_CLIENT_AUTH](#)

To determine the details behind the client authentication used for a particular user session, query the following tables in the V_MONITOR schema:

- [SESSIONS](#)
- [USER_SESSIONS](#)

Enabling and Disabling Authentication Methods

When you create an authentication method, Vertica stores it in the catalog and enables it automatically. To enable or disable an authentication method, use the [ALTER AUTHENTICATION](#) statement. To use this approach, you must be connected to your database.

If an authentication method has not been enabled, Vertica cannot use it to authenticate users and clients trying to connect to the database.

To enable an authentication method:

```
ALTER AUTHENTICATION v_kerberos ENABLE;
```

To disable this authentication method:

```
ALTER AUTHENTICATION v_kerberos DISABLE;
```

See Also

- [Creating Authentication Records](#)
- [Deleting Authentication Records](#)
- [Granting and Revoking Authentication Methods](#)
- [Modifying Authentication Records](#)

Granting and Revoking Authentication Methods

Before Vertica can validate a user or client through an authentication method, you must first associate that authentication method with the user or role that requires it. To do this, use `GRANT AUTHENTICATION`. When that user or role no longer needs to connect to Vertica using that method, you can disassociate that authentication from that user with `REVOKE AUTHENTICATION`.

Grant Authentication Methods

You can grant an authentication method to a specific user or role. You can also specify the default authentication method by granting an authentication method to `Public`. Use the [GRANT \(Authentication\)](#) statement as follows:

This example uses a `GRANT AUTHENTICATION` statement to associate `v_ldap` authentication with user `jsmith`:

```
=> GRANT AUTHENTICATION v_ldap TO jsmith;
```

This example uses a `GRANT AUTHENTICATION` statement to associate `v_gss` authentication to the role `DBprogrammer`:

```
=> CREATE ROLE DBprogrammer;  
=> GRANT AUTHENTICATION v_gss to DBprogrammer;
```

This example sets the default client authentication method to `v_localpwd`:

```
=> GRANT AUTHENTICATION v_localpwd TO Public;
```

Revoke Authentication Methods

If you no longer want to authenticate a user or client with a given authentication method, use the [REVOKE \(Authentication\)](#) statement as follows:

This example revokes `v_ldap` authentication from user `jsmith`:

```
=> REVOKE AUTHENTICATION v_ldap FROM jsmith;
```

This example revokes `v_gss` authentication from the role `DBprogrammer`:

```
=> REVOKE AUTHENTICATION v_gss FROM DBprogrammer;
```

This example removes `localpwd` as the default client authentication method:

```
=> REVOKE AUTHENTICATION localpwd from Public;
```

Hash Authentication

Vertica provides a hash authentication method that allows you to use the MD5 algorithm or the more secure algorithm, SHA-512, to store user passwords. SHA-512 is one of the industry-standard SHA-2 family of hash algorithms that address weaknesses in SHA-1 and MD5. For information on creating passwords to be hashed during authentication see [Passwords](#)

Note: Micro Focus strongly recommends that you use SHA-512 for hash authentication because it is more secure than MD5.

Before you perform hash authentication, review the following topics:

- [Hash Authentication Parameters](#)—Describes the two hash authentication parameters that specify which hashing algorithm to use.

- [Upgrade Considerations for Hash Authentication](#)—Before you implement the SHA-512 algorithm for one or more users, you must be aware of several issues. For details, review this topic before proceeding.
- [How to Configure Hash Authentication](#)—After you decide to implement hash authentication in your database, follow the steps described in this topic..

Hash Authentication Parameters

Two parameters control which algorithm hash authentication uses for hashing and storing user passwords:

- A user-level parameter, `Security_Algorithm`:

```
=> ALTER USER username SECURITY_ALGORITHM 'MD5' IDENTIFIED BY 'newpassword';  
=> ALTER USER username SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';
```

- A system-level configuration parameter, `SecurityAlgorithm`:

```
=> SELECT SET_CONFIG_PARAMETER('SecurityAlgorithm', 'MD5');  
=> SELECT SET_CONFIG_PARAMETER('SecurityAlgorithm', 'SHA512');
```

Both parameters can have the following values:

- 'NONE'
- 'MD5'
- 'SHA512'

Note: If your current password is in the MD5 format you cannot rename a user with [ALTER USER](#).

The user-level parameter usually has precedence over the system-level parameter. However, if the user-level parameter is 'NONE', Vertica hashes passwords with the algorithm assigned to the system-level parameter value. If both parameters are 'NONE', Vertica uses the MD5 algorithm.

These values, which are stored in the [PASSWORD_AUDITOR](#) system table, affect the security algorithm that is actually used for hash authentication.

User-Level Parameter Value	System-Level Parameter Value	Algorithm Used for Hash Authentication	Algorithm Used for Hash Authentication - FIPS mode
'NONE'	'NONE'	MD5	SHA-512
'NONE'	'MD5'	MD5	SHA-512
'NONE'	'SHA512'	SHA-512	SHA-512
'MD5'	'NONE'	MD5	SHA-512
'MD5'	'MD5'	MD5	SHA-512
'MD5'	'SHA512'	MD5	SHA-512
'SHA512'	'NONE'	SHA-512	SHA-512
'SHA512'	'MD5'	SHA-512	SHA-512
'SHA512'	'SHA512'	SHA-512	SHA-512

How to Configure Hash Authentication

Follow these steps to configure hash authentication:

1. Create an authentication method that is based on hash encryption. When you create an authentication method, it is automatically enabled for use.

The following example shows how to create an authentication method `v_hash` for users logging in from the IP address `10.0.0.0/0`.

```
=> CREATE AUTHENTICATION v_hash METHOD 'hash' HOST '10.0.0.0/0';
```

If users are trying to connect from an IPv6 address, the statement might look like this example:

```
=> CREATE AUTHENTICATION v_hash METHOD 'hash' HOST '2001:db8:ab::123/128';
```

2. Decide which password-hashing algorithm you want to use: MD5 or the more secure SHA-512.
3. Specify the security algorithm as follows:

- At the system level, set the **SecurityAlgorithm** configuration parameter. This setting applies to all users, unless their user-level security is set to another value:

```
=> ALTER DATABASE mydb SET SecurityAlgorithm = 'MD5';  
=> ALTER DATABASE mydb SET SecurityAlgorithm = 'SHA512';
```

If you want users to inherit the system-level security, set their passwords to expire immediately. Users must change their passwords before they log in again. Alternatively, you can ask users to change their passwords. Vertica hashes all new passwords using the system-level security algorithm.

- At the user level, use **ALTER USER** to set the `Security_Algorithm` user parameter. Changing this parameter at the user level overrides the system-level value:

```
=> ALTER USER username SECURITY_ALGORITHM 'MD5' IDENTIFIED BY 'newpassword';  
=> ALTER USER username SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';
```

4. Associate the `v_hash` authentication method with the desired users or user roles, using a **GRANT** statement:

```
=> GRANT AUTHENTICATION v_hash to user1, user2, ...;
```

For more information about how these parameters work, see [Hash Authentication Parameters](#).

Upgrade Considerations for Hash Authentication

For Vertica releases before 7.1, MD5 is the only algorithm used for hashing passwords. In Vertica 7.1, you can use either the MD5 algorithm or the more secure SHA-512 algorithm. Before you upgrade, you must consider the following behaviors to avoid problems.

Upgrade the Client and Server

To implement the more secure SHA-512 algorithm for hashing passwords, you *must* upgrade BOTH the client and the server to Vertica 7.1 or higher. If you upgrade the server but not the client and specify that one or more users store their passwords using SHA-512, the client does not understand hashing with SHA-512. When it sends a message to the server, the server returns an error.

Change Existing Users to SHA-512 Hash Algorithm

When you upgrade from a pre-7.1 database, the user-level parameter `Security_Algorithm`, is set to `'NONE'`. This allows all existing users to continue connecting to the Vertica server and their passwords are hashed using MD5.

If you want one or more users to use the SHA-512 algorithm, set the system-level parameter `Security Algorithm` to `'SHA512'` and change the user passwords.

Use one of three methods to change the user password:

- Manually set the user's user-level security algorithm to `'SHA512'`. Then, change the user's password, as in the following statement:

```
=> ALTER USER username SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';
```

- Set the user's password to expire immediately as in the following statement. After the password expires, the user responds by changing it.

```
=> ALTER USER username PASSWORD EXPIRE;
```

- Ask the user to change the password.

All new passwords inherit the system-level security algorithm, which is SHA-512.

Passwords

Assign a password to a user to allow that user to connect to the database using password authentication. When the user supplies the correct password a connection to the database occurs.

Vertica stores passwords in an encrypted format to prevent potential theft. However, the transmission of the password to Vertica is in plain text. Thus, it is possible for a "man-in-the-middle" attack to intercept the password.

Implementing [Hash Authentication](#) ensures secure login using passwords.

About Password Creation and Modification

You must be a superuser to create passwords for user accounts using the [CREATE USER](#) statement. A superuser can set any user account's password.

- To add a password, use the [ALTER USER](#) statement.
- To change a password, use [ALTER USER](#) or the vsql `\password` command.

Users can also change their own passwords.

To make password authentication more effective, Vertica recommends that you enforce password policies that control how often users are forced to change passwords and the required content of a password. You set these policies using [Profiles](#).

Default Password Authentication

When you have not specified any authentication methods, Vertica defaults to using password authentication for user accounts that have passwords.

If you create authentication methods, even for remote hosts, password authentication is disabled. In such cases, you must explicitly enable password authentication. The following commands create the `local_pwd` authentication method and make it the default for all users. When you create an authentication method, Vertica enables it automatically:

```
=> CREATE AUTHENTICATION local_pwd METHOD hash' LOCAL;  
=> GRANT AUTHENTICATION local_pwd To Public;
```

Profiles

You set password policies using profiles. A *profile* is a group of parameters that includes requirements for user passwords. A profile controls:

- How often users must change their passwords.
- How many times users must change their passwords before they can reuse an old password.
- How many times a user can fail to log in before the account is locked.
- The required length and content of the password:
 - Maximum and minimum number of characters
 - Minimum number of capital letters, lowercase letters, digits, and symbols required in a password.

To set a user's password policy, assign the user to a profile. To enforce different password policies for different users, create multiple profiles. For example, you might create one profile

for interactive users, requiring them to frequently change their passwords. You might create another profile for user accounts that are not required to change passwords.

Create and Modify Profiles

You create profiles using the [CREATE PROFILE](#) statement and change profiles using [ALTER PROFILE](#). You can assign a user to a profile when you create the user ([CREATE USER](#)), or after, using the [ALTER USER](#) statement. A user can be assigned to only one profile at a time.

All newly created databases contain an initial profile named DEFAULT. Vertica assigns all users to the DEFAULT profile if:

- You do not explicitly assign users a profile when you create them.
- You drop the profile to which a user is currently assigned.

You can change the policy parameters in the DEFAULT profile, but you cannot delete it.

Important: During upgrades from versions of Vertica earlier than version 5.0, each database receives a DEFAULT profile. All users are then automatically assigned to that profile.

The profiles that you create can inherit some or all of their policy parameters from the DEFAULT profile. When you create a profile using [CREATE PROFILE](#), a parameter inherits its value from the DEFAULT profile if:

- You set it to the DEFAULT value.
- You do not assign a value.

If you change a parameter in the DEFAULT profile, you also change that parameter's value in every profile that inherits the parameter from DEFAULT.

Changes to a profile's policies for password content do not have an immediate effect on the users. When Vertica does not test user's passwords to verify that they comply with the new password criteria. Instead, the changed settings only affect the users the next time they change their password. To make sure that users comply with the new password policy, use the [ALTER USER](#) statement to expire user passwords. Vertica prompts users with expired passwords to change their passwords when they next log in.

Note: Only the profile settings for how many failed login attempts trigger [Account Locking](#). All password [complexity](#), reuse, and [lifetime settings](#) affect only passwords that Vertica manages.

See Also

- [PROFILES](#)
- [Creating a Database Name and Password](#)

Password Guidelines

For passwords to be effective, they must be hard to guess. You need to protect passwords from:

- Dictionary-style, brute-force attacks
- Users who have knowledge of the password holder (family names, birth dates , etc.)

Use [Profiles](#) to enforce good password practices (password length and required content). Make sure database users know the password guidelines, and encourage them not to use personal information in their passwords.

For guidelines on creating strong passwords go to [Microsoft Tips for Creating a Strong Password](#).

See Also

- [Creating a Database Name and Password](#)

Password Expiration

User profiles control how often users must change their passwords. Initially, the DEFAULT profile is set so that passwords never expire.

Important: Password expiration has no effect on any of the user's current sessions.

Set Password Expiration and Grace Period

You can change the default value to set a password expiration. Alternatively, you can create additional profiles that set time limits for passwords and assign users to them.

When a password expires, the user must change the password on the next login. However, you can set a PASSWORD_GRACE_TIME in any individual user's profile, allowing that user to log in

after the expiration. After the password expires, Vertica issues a warning about the password expiration but continues to recognize the password.

After the grace period ends, users must change their passwords to log in, unless they have changed them already in response to the warning.

Expire a Password

You can expire a user's password immediately using the [ALTER USER](#) statement's `PASSWORD EXPIRE` parameter. By expiring a password, you can:

- Force users to comply with a change to password policy.
- Set a new password when a user forgets the old password.

Account Locking

In a profile, you can set a password policy for how many consecutive failed login attempts a user account is allowed before locking. This locking mechanism helps prevent dictionary-style brute-force attempts to guess users' passwords.

Set Account Locking

Set this value using the `FAILED_LOGIN_ATTEMPTS` parameter using the [CREATE PROFILE](#) or [ALTER PROFILE](#) statement.

Vertica locks any user account that has more consecutive failed login attempts than the value to which you set `FAILED_LOGIN_ATTEMPTS`. The user cannot log in to a locked account, even by supplying the correct password.

Unlock a Locked Account

You can unlock accounts in one of two ways, depending on your privileges.

- **Manually**—If you are a superuser, you can manually unlock the account using the [ALTER USER](#) command.

Note: A superuser account cannot be locked, because it is the only user that can unlock accounts. For this reason, choose a very secure password for a superuser account. See [Password Guidelines](#) for suggestions.

- **Password Lock Time Setting**—Specify the number of days until an account unlocks in the `PASSWORD_LOCK_TIME` parameter of the user's profile. Vertica automatically unlocks the account after the specified number of days has passed. If you set this parameter to `UNLIMITED`, the user's account is never automatically unlocked, and a superuser must manually unlock it.

Ident Authentication

The Ident protocol, defined in [RFC 1413](#), authenticates a database user with a system user name. To see if that system user can log in without specifying a password, you configure Vertica client authentication to query an Ident server. With this feature, the `DBADMIN` user can run automated scripts to execute tasks on the Vertica server.

Caution: Ident responses can be easily spoofed by untrusted servers. Use Ident authentication only on local connections, where the Ident server is installed on the same computer as the Vertica database server.

Following the instructions in these topics to install, set up, and configure Ident authentication for your database:

- [Installing and Setting Up an Ident Server](#)
- [Configuring Ident Authentication for Database Users](#)

Examples

The following examples show several ways to configure Ident authentication.

Allow `system_user1` to connect to the database as Vertica `vuser1`:

```
=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION v_ident SET system_users='system_user1';  
=> GRANT AUTHENTICATION v_ident to vuser1;  
=> ALTER AUTHENTICATION v_ident ENABLE;
```

Allow `system_user1`, `system_user2`, and `system_user3` to connect to the database as `vuser1`. Use colons (`:`) to separate the user names:

```
=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION v_ident SET system_users='system_user1:system_user2:system_user3';  
=> GRANT AUTHENTICATION v_ident TO vuser1;  
=> ALTER AUTHENTICATION v_ident ENABLE;
```

Associate the authentication with `Public` using a `GRANT AUTHENTICATION` statement. The users, `system_user1`, `system_user2`, and `system_user3` can now connect to the database as any database user:

```
=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION v_ident SET system_users='system_user1:system_user2:system_user3';  
=> GRANT AUTHENTICATION v_ident to Public;  
=> ALTER AUTHENTICATION v_ident ENABLE;
```

Set the `system_users` parameter to `*` to allow any system user to connect to the database as `vuser1`:

```
=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION v_ident SET system_users='*';  
=> GRANT AUTHENTICATION v_ident TO vuser1;  
=> ALTER AUTHENTICATION v_ident ENABLE;
```

Using a `GRANT` statement, associate the `v_ident` authentication with `Public` to allow `system_user1` to log into the database as any database user:

```
=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;  
=> ALTER AUTHENTICATION v_ident SET system_users='system_user1';  
=> GRANT AUTHENTICATION v_ident to Public;  
=> ALTER AUTHENTICATION v_ident ENABLE;
```

Installing and Setting Up an Ident Server

To use Ident authentication, you must install one or more packages, depending on your operating system, and enable the Ident server on your Vertica server. `oidentd` is an Ident daemon that is compatible with Vertica and compliant with [RFC 1413](#).

Note: You can find the source code and installation instructions for `oidentd` at the [oidentd website](#).

To install and configure Ident authentication for use with your Vertica database, follow the appropriate steps for your operating system:

Red Hat 6.x/CentOS 6.x

Install `oidentd` on Red Hat 6.x or CentOS 6.x by running this command:

```
$ yum install oidentd
```

Depending on your configuration, you might receive the following error message:

No package `oidentd` available.

In this case, you must install the Red Hat/CentOS Extras Repository. Download and install the Extras Repository from the following location: <https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm>

Red Hat 7.x/CentOS 7.x

Install an Ident server on Red Hat 7.x or CentOS 7.x by installing the `authd` and `xinetd` packages:

```
$ yum install authd
$ yum install xinetd
```

Ubuntu/Debian

Install `oidentd` on Ubuntu or Debian by running this command:

```
$ sudo apt-get install oidentd
```

SUSE Linux Enterprise Server

Install the `pidentd` and `xinetd` RPMs from the following locations:

- https://www.suse.com/LinuxPackages/packageRouter.jsp?product=server&version=11&service_pack=&architecture=i386&package_name=pidentd
- https://www.suse.com/LinuxPackages/packageRouter.jsp?product=server&version=11&service_pack=&architecture=i386&package_name=xinetd

Post-Installation Steps for Red Hat 6.x/CentOS 6.x and Ubuntu/Debian

After you install `oidentd` on your Red Hat 6.x/CentOS 6.x or Ubuntu/Debian system, continue with the following steps:

1. Verify that the Ident server accepts IPv6 connections to prevent authentication failure. To do so, you must enable this capability. In the script `/etc/init.d/oidentd`, change the line from:

```
exec="/usr/sbin/oidentd"
```

to

```
exec="/usr/sbin/oidentd -a ::"
```

Then, at the Linux prompt, start `oidentd` with `-a ::`.

2. Restart the server with the following command:

```
$ /etc/init.d/oidentd restart
```

Post-Installation Steps for Red Hat 7.x/CentOS 7.x and SUSE Linux Enterprise Server

After you install the required packages on your Red Hat 7.x/CentOS 7.x or SUSE Linux Enterprise Server system, continue with the following steps:

1. Enable the `auth` service in the configuration file located at the following location:
`/etc/xinet.d/auth`.

Enter `no` for the `disable` option, as this sample configuration file shows.

```
service auth
{
    disable = no
    socket_type = stream
    wait = no
    user = ident
    cps = 4096 10
    instances = UNLIMITED
    server = /usr/sbin/in.authd
    server_args = -t60 --xerror --os
}
```

2. Restart the `xinetd` service with the following command:

```
$ service xinetd restart
```

Configuring Ident Authentication for Database Users

To configure Ident authentication, take the following steps:

1. Create an authentication method that uses Ident.

The Ident server must be installed on the same computer as your database, so specify the keyword LOCAL. Vertica requires that the Ident server and database always be on the same computer as the database.

```
=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
```

2. Set the Ident authentication parameters, specifying the system users who should be allowed to connect to your database.

```
=> ALTER AUTHENTICATION v_ident SET system_users='user1:user2:user3';
```

3. Associate the authentication method with the Vertica user. Use a GRANT statement that allows the system user user1 to log in using Ident authentication:

```
=> GRANT AUTHENTICATION v_ident TO user1;
```

Kerberos Authentication

Kerberos authentication uses the following

Client Package

The Kerberos 5 client package communicates with the KDC server. This package is not included as part of the Vertica Analytics Platform installation. Kerberos software is built into Microsoft Windows. If you are using another operating system, you must obtain and install the client package.

If you do not already have the Kerberos 5 client package on your system, download it from the [MIT Kerberos Distribution page](#). Install the package on each Vertica server and client used in Kerberos authentication, except the KDC itself.

Refer to the [Kerberos documentation](#) for installation instructions.

Service Principals

A service principal consists of a host name and a realm to which a set of credentials gets assigned. These credentials connect to the service which is a host that you connect to over your network and authenticate using the KDC. .

See [Specify KDC Information and Configure Realms](#) to create the realm name. The host name must match the value supplied by the operating system. Typically this is the fully qualified host name. If the host name part of your principal does not match the value supplied by the operating system, Kerberos authentication fails.

Some systems use a hosts file (/etc/hosts or /etc/hostnames) to define host names. A hosts file can define more than one name for a host. The operating system supplies the first entry, so use that in your principal. For example, if your hosts file contains:

```
192.168.1.101 v_vmart_node0001.example.com v_vmart_node0001
```

then use v_vmart_node0001.example.com as the hostname value.

Note: Depending on your configuration it may be safer to use the fully qualified domain name rather than the hostname.

Configure the following as Kerberos principals:

- Each client (users or applications that connects to Vertica)
- The Vertica server

See the following topics for more information:

- [Configure Vertica for Kerberos Authentication](#)
- [Configure Clients for Kerberos Authentication](#)

Keytab Files

Principals are stored in encrypted keytab files. The keytab file contains the credentials for the Vertica principal. The keytab allows the Vertica server to authenticate itself to the KDC. You need the keytab so that Vertica Analytic Database does not have to prompt for a password.

Create one service principal for each node in your cluster. You can then either create individual keytab files (one for each node containing only that node's principal) or create one keytab file containing all the principals.

- **Create one keytab file with all principals** to simplify setup: all nodes have the same file, making initial setup easier. If you add nodes later you either update (and redistribute) the global keytab file or make separate keytabs for the new nodes. If a principal is compromised it is compromised on all nodes where it is present in a keytab file.
- **Create separate keytab files on each node** to simplify maintenance. Initial setup is more involved as you must create a different file on each node, but no principals are shared across nodes. If you add nodes later you create keytabs on the new nodes. Each node's keytab contains only one principal, the one to use for that node.

Ticket-Granting Ticket

The Ticket-Granting Ticket (TGT) retrieves service tickets that authenticates users to servers in the domain. Future login requests use the cached HTTP Service Ticket for authentication, unless it has expired as set in the `ticket_lifetime` parameter in `krb5.conf`.

Configure Vertica for Kerberos Authentication

Kerberos provides a strong cryptographic authentication against the devices which lets the client & servers to communicate in a more secured manner. It addresses network security problems.

Your system must have one or more Kerberos Key Distribution Centers (KDC) installed and configured. The KDCs must be accessible from every node in your Vertica Analytics Platform cluster.

The KDC must support Kerberos 5 using GSS-API. For details, see the [MIT Kerberos Distribution Page](#).

In This Section

Create the Vertica Principals and Keytabs on Linux KDC

Vertica uses service principals for system-level operations. These principals identify the Vertica service and are used as follows:

- Kerberized Vertica clients request access to this service when they authenticate to the database.
- System processes like the Tuple Mover use this identity when they authenticate to external services such as Hadoop.

Create principals and keys as follows:

1. Start the Kerberos 5 database administration utility (`kadmin` or `kadmin.local`) to create Vertica principals on a Linux KDC.
 - Use `kadmin` if you are accessing the KDC on a remote server. If you have access to the Kerberos administrator password, you can use `kadmin` on any machine where the Kerberos 5 client package is installed. When you start `kadmin`, the utility prompts you for the Kerberos administrator's password. You might need root privileges on the client to run `kadmin`.
 - Use `kadmin.local` if:
 - The KDC is on the machine that you are logging in to.
 - You have root privileges on that server.

`kadmin.local` does not require the administrators login credentials.

For more information about the `kadmin` and `kadmin.local` commands, see the [kadmin documentation](#).

2. Create one service principal for Vertica on each node. The host name must match the value supplied by the operating system. The following example creates the service principal `vertica` for the node named `v_vmart_node0001.example.com`:

```
$ sudo /usr/kerberos/sbin/kadmin.local  
kadmin.local add_principal vertica/v_vmart_node0001.example.com
```

Repeat the `ktadd` command once per principal. You can create separate keytabs for each principal user or add them all to a single keytab file (such as `krb5.keytab`). If you are using a single file, see the documentation for the `-glob` option in the [MIT Kerberos documentation](#).

You must have a user principal for each Vertica Analytics Platform user that uses Kerberos Authentication. For example:

```
$ sudo add_principal [options] VerticaUser1
```

3. Copy each keytab file to the /etc folder on the corresponding cluster node. Use the same path and file name on all nodes.
4. On each node, make the keytab file readable by the file owner who is running the database process (typically, the Linux dbadmin user). For example, you can change ownership of the files to dbadmin as follows:

```
$ sudo chown dbadmin *.keytab
```

Important: In a production environment, you must control who can access the keytab file to prevent unauthorized users from delegating your server. For more information about delegation (also known as impersonation), see [Technet.Microsoft.com](https://technet.microsoft.com).

After you create a keytab file, you can use the `klist` command to view keys stored in the file:

```
$ sudo /usr/kerberos/bin/klist -ke -t
Keytab name: FILE:/etc/krb5.keytab
KVNO Principal
-----
4 vertica/v_vmart_node0001.example.com@EXAMPLE.COM
4 vertica/v_vmart_node0001.example.com@EXAMPLE.COM
```

5. On Vertica run the following to ensure the Kerberos parameters are set correctly:

```
=> select parameter_name, current_value from vs_configuration_parameters where
parameter_name like 'Ker%' or parameter_name like 'ClientAuthentication';
parameter_name | current_value
-----+-----
ClientAuthentication | host kuser 0.0.0.0/0 gss, host nokrb 0.0.0.0/0 md5, local all trust
KerberosHostname | v_vmart_node0001.example.com
KerberosKeytabFile | /scratch_b/qa/krbTest/v_krbtest_node0001_catalog/krb5.keytab
KerberosRealm | EXAMPLE.COM
KerberosServiceName | vertica
(5 rows)
```

6. On Vertica, run `KERBEROS_CONFIG_CHECK` to verify the Kerberos configuration. `KERBEROS_CONFIG_CHECK` verifies the following:
 - The existence of the `kinit` and `kb5.conf` files.
 - Whether the keytab file exists and is set

- The Kerberos configuration parameters set in the database:
 - KerberosServiceName
 - KerberosHostname
 - KerberosRealm
 - Vertica Principal
- That Kerberos can read the Vertica keys
- That Kerberos can get the tickets for the Vertica principal
- That Vertica can initialize the keys with kinit

Creating the Principals and Keytab on Active Directory

Active Directory stores information about members of the Windows domain, including users and hosts.

Vertica uses the Kerberos protocol to access this information in order to authenticate windows users to the vertica database. The Kerberos protocol uses principals to identify users and keytab files to store their cryptographic information. You need to install the keytab files into Vertica to enable the Vertica database to cryptographically authenticate windows users.

This procedure describes :

- The creation of a Vertica service principal.
 - Exporting the keytab files for these principals
 - Installing the keytab files in the Vertica database. This allows Vertica to authenticate windows users and grant them access to the Vertica database.
1. Create a Windows account (principal) for the Vertica service and one Vertica host for each node/host in the cluster. This procedure creates windows accounts for host `verticanode01` and service `vertica` running on this node.

When you create these accounts, select the following:

- User Cannot change password
- Password never expires

Note: You can deselect **Password never expires**, but if you change these user passwords you must recreate the keytab files and reinstall them into Vertica. This includes repeating the entire procedure.

2. If you are using external tables on HDFS that are secured by kerberos authentication, you **MUST** enable Delegation. To do so, access the Active Directory Users and Computers dialog, right-click on the Windows account (principal) for the Vertica service, and select Delegation. Trust this user for delegation to any service.
3. Run the following command to create the keytab for the host `verticanode01.dc.com` node/host:

```
$ ktpass -out ./host.verticanode01.dc.com.keytab -princ host/verticanode01.dc.com@DC.COM -  
mapuser verticanode01  
-mapop set -pass secret -ptype KRB5_NT_SRV_HST
```

4. Run the following command to create the keytab for the vertica service:

```
$ ktpass -out ./vertica.verticanode01dc.com.keytab -princ vertica/verticanode01.dc.com@DC.COM -  
mapuser vertica  
-mapop set -pass secret -ptype KRB5_NT_PRINCIPAL
```

For more information about keytab files, see Technet.Microsoft.com.

5. Run the following commands to ensure the service principal name is mapped correctly. You must run these commands for each node in your cluster:

```
$ setspn -L vertica  
Registered ServicePrincipalNamefor CN=vertica,CN=Users,DC=dc,DC=com  
vertica/verticanode01.dc.com  
  
$ setspn -L verticanode01  
Registered ServicePrincipalNamefor CN=verticanode01,CN=Users,DC=dc,DC=com  
host/verticanode01.dc.com
```

6. Copy the keytabs you created above, `vertica.verticanode01.dc.com.keytab` and `host.verticanode01.dc.com.keytab`, to the Linux host `verticanode01.dc.com`.
7. Combine the keytab files into a single keytab:

```
[release@vertica krbTest]$ /usr/kerberos/sbin/ktutil  
ktutil: rkt host.verticanode01.dc.com.keytab  
ktutil: rkt vertica.verticanode01.dc.com.keytab  
ktutil: list  
slot KVNO Principal  
-----
```

```
1 3 host/verticanode01.dc.com@DC.COM
2 16 vertica/verticanode01.dc.com@DC.COM
ktutil: wkt verticanode01.dc.com.keytab
ktutil: exit
```

This creates a single keytab file that contains the server principal for authentication.

8. Copy the new keytab file to the catalog directory. For example:

```
$ cp verticanode01.dc.com.keytab /home/dbadmin/VMart/v_vmart_nodennnn_catalog
```

9. Test the keytab file's ability to retrieve a ticket to ensure it works from the Vertica node:

```
$ kinit vertica/verticanode01.dc.com -k -t verticanode01.dc.com.keytab
$ klist

Ticket cache: KFILE:/tmp/krb_ccache_1003
Default principal: vertica/verticanode01.dc.com@DC.COM

Valid starting Expires Service principal
04/08/2017 13:35:25 04/08/2017 23:35:25 krbtgt/DC.COM@DC.COM
renew until 04/15/2017 14:35:25
```

When the ticket expires or not automatically retrieved you need to manually run the kinit command. See [Get the Kerberos Ticket and Authenticate Vertica](#) .

10. Set the right permissions and ownership on the keytab files:

```
$ chmod 600 verticanode01.dc.com.keytab
$ chown dbadmin:verticadba verticanode01.dc.com.keytab
```

11. Set the following [Kerberos Authentication Parameters](#) using `ALTER DATABASE` to inform Vertica about the Kerberos principal:

```
KerberosKeytabFile=<CATALOGDIR>/verticanode01.dc.com.keytab
KerberosRealm=DC.COM
KerberosServiceName=vertica
KerberosTicketDuration = 0
KerberosHostname=verticanode01.dc.com
```

12. Restart the Vertica server.
13. Test your Kerberos setup as follows to ensure that all clients use the gss authentication method.

From Vertica:

```
=> CREATE USER windowsuser1;
CREATE USER

=> CREATE AUTHENTICATION v_kerberos method 'gss' host '0.0.0.0/0';
CREATE AUTHENTICATION

=> ALTER AUTHENTICATION v_kerberos enable;
ALTER AUTHENTICATION

=> GRANT AUTHENTICATION v_kerberos to windowsuser1;
GRANT AUTHENTICATION
```

From OS command line:

```
$ kinit windowsuser1

$ vsql -U windowsuser1 -k vertica -K verticanode01.dc.com -h verticanode01.dc.com -c "select
client_authentication_name,
authentication_method from sessions;"
client_authentication_name | authentication_method
-----+-----
v_kerberos                  |      GSS-Kerberos
(1 row)
```

14. Run [KERBEROS_CONFIG_CHECK](#) to verify the Kerberos configuration. `KERBEROS_CONFIG_CHECK` verifies the following:
 - The existence of the `kinit` and `kb5.conf` files.
 - Whether the keytab file exists and is set
 - The Kerberos configuration parameters set in the database:
 - `KerberosServiceName`
 - `KerberosHostname`
 - `KerberosRealm`
 - `Vertica Principal`
 - That Kerberos can read the Vertica keys
 - That Kerberos can get the tickets for the Vertica principal
 - That Vertica can initialize the keys with `kinit`

Get the Kerberos Ticket and Authenticate Vertica

If your organization uses Kerberos as part of the login process, Kerberos tickets are automatically retrieved upon login. Otherwise, you need to run `kinit` to retrieve the Kerberos ticket.

The following example shows how to retrieve the ticket and authenticate Vertica Analytics Platform with the KDC using the `kinit` command. `EXAMPLE.COM` is the realm name. You must use the realm name with your username to retrieve a Kerberos ticket. See [Specify KDC Information and Configure Realms](#).

```
$ kinit  
Password for principal_user@EXAMPLE.COM: kpasswd
```

You are prompted for the password of the principal user name created when you created the principals and keytabs (see [Create the Vertica Principals and Keytabs on Linux KDC](#)).

The Kerberos ticket gets cached for a pre-determined length of time. See [Ticket Management](#) in the Kerberos documentation for more information on setting expiration parameters.

Upon expiration, you need to run the `kinit` command again to retrieve another Kerberos ticket.

Configure Clients for Kerberos Authentication

Each supported platform has a different security framework. Thus, the steps required to configure and authenticate against Kerberos differ among clients.

On the server side, you construct the Vertica Kerberos service name principal using this format:

```
Kerberos_Service_Name/Kerberos_Host_Name@Kerberos_Realm
```

For each client, the GSS libraries require the following format for the Vertica service principal:

```
Kerberos_Service_Name@Kerberos_Host_Name
```

You can omit the realm portion of the principal because GSS libraries use the realm name of the configured default (*Kerberos_Realm*) realm.

For information about client connection strings, see the following topics in [Connecting to Vertica](#):

- [ODBC DSN Parameters](#)
- [JDBC Connection Properties](#)
- [ADO.NET Connection Properties](#)
- (vsq) [Command-Line Options](#)

Note: A few scenarios exist in which the Vertica server principal name might not match the host name in the connection string. See [Troubleshooting Kerberos Authentication](#) for more information.

In This Section

- [Configure ODBC and vsq Clients on Linux, HP-UX, AIX, MAC OSX, and Solaris](#)
- [Configure ODBC and vsq Clients on Windows and ADO.NET](#)
- [Configure JDBC Clients on all Platforms](#)

Configure ODBC and vsq Clients on Non-Windows Platforms

To configure an ODBC or vsq client on Linux, HP-UX, AIX, MAC OSX, or Solaris, you must first install the Kerberos 5 client package. See [Configuring Kerberos Authentication](#).

After you install the Kerberos 5 client package, you must provide clients with a valid Kerberos configuration file (krb5.conf). To communicate with the KDC, each client participating in Kerberos authentication must have a valid, identically configured krb5.conf file. The default location for the Kerberos configuration file is /etc/krb5.conf.

Tip: To enforce consistency among clients, Vertica Analytic Database, and the KDC, copy the /etc/krb5.conf file from the KDC to the client's/etc directory.

The Kerberos configuration (krb5.conf) file contains Kerberos-specific information, including:

- How to reach the KDC
- Default realm name
- Domain
- Path to log files

- DNS lookup
- Encryption types to use
- Ticket lifetime

The default location for the Kerberos configuration file is `/etc/krb5.conf`.

When configured properly, the client can authenticate with Kerberos and retrieve a ticket through the `kinit` utility (see [Acquire an ODBC Authentication Request and Connection](#) below). Likewise, the server can then use `ktutil` to store its credentials in a keytab file

Authenticating ODBC and vsql Clients Requests and Connections on Non-Windows Platforms

ODBC and `vsql` use the client's ticket established by `kinit` to perform Kerberos authentication. These clients rely on the security library's default mechanisms to find the ticket file and the and Kerberos configuration file.

To authenticate against Kerberos, call the `kinit` utility to obtain a ticket from the Kerberos KDC server. The following two examples show how to send the ticket request using ODBC and `vsql` clients.

Acquire an ODBC Authentication Request and Connection

1. On an ODBC client, acquire a ticket for the `kuser` user by calling the `kinit` utility.

```
$ kinit kuser@EXAMPLE.COM
Password for kuser@EXAMPLE.COM:
```

2. Connect to Vertica, and provide the principals in the connection string:

```
char outStr[100];
SQLLEN len;
SQLDriverConnect(handle, NULL, "Database=VMart;User=kuser;
Server=myserver.example.com;Port=5433;KerberosHostname=vcluster.example.com",
SQL_NTS, outStr, &len);
```

Acquire a vsql Authentication Request Connection

If the `vsql` client is on the same machine you are connecting to, `vsql` connects through a UNIX domain socket. This connection bypasses Kerberos authentication. When you authenticate

with Kerberos, especially if the client authentication method is configured as 'local', you must include the `-h` hostname option. See [Command Line Options](#) in Connecting to Vertica.

1. On the vsql client, call the `kinit` utility:

```
$ kinit kuser@EXAMPLE.COM
Password for kuser@EXAMPLE.COM:
```

2. Connect to Vertica, and provide the host and user principals in the connection string:

```
$ ./vsql -K vcluster.example.com -h myserver.example.com -U kuser
Welcome to vsql, the Vertica Analytic Database
interactive terminal.

Type: \h or \? for help with vsql commands
\g or terminate with semicolon to execute query
\q to quit
```

In the future, when you log in to vsql as kuser, vsql uses your cached ticket without prompting you for a password.

Verify the Authentication Method

You can verify the authentication method by querying the `SESSIONS` system table:

```
=> SELECT authentication_method FROM sessions;
 authentication_method
-----
GSS-Kerberos
(1 row)
```

See Also

- [Data Source Name \(DSN\) Connection Properties](#) in Connecting to Vertica
- (vsql) [Command-Line Options](#) in Connecting to Vertica

Configure ADO.NET, ODBC, and vsql Clients on Windows

The Vertica client drivers support the Windows SSPI library for Kerberos authentication. Windows Kerberos configuration is stored in the registry.

You can choose between two different setup scenarios for Kerberos authentication on ODBC and vsql clients on Windows and ADO.NET:

- [Windows KDC on Active Directory with Windows Built-in Kerberos Client and Vertica](#)
- [Linux KDC with Windows Built-in Kerberos Client and Vertica](#)

Windows KDC on Active Directory with Windows Built-in Kerberos Client and Vertica

Kerberos authentication on Windows is commonly used with Active Directory, Microsoft's enterprise directory service/Kerberos implementation. Typically your organization's network or IT administrator performs the setup.

Windows clients have Kerberos authentication built into the authentication process. You do not need any additional software.

Your login credentials authenticate you to the Kerberos server (KDC) when you:

- Log in to Windows from a client machine
- Use a Windows instance that has been configured to use Kerberos through Active Directory

To use Kerberos authentication on Windows clients, log in as REALM\user.

Important: When you use the ADO.NET driver to connect to Vertica, you can optionally specify `IntegratedSecurity=true` in the connection string. This informs the driver to authenticate the calling user against the user's Windows credentials. As a result, you do not need to include a user name or password in the connection string. Any `user=<username>` entry to the connection string is ignored.

Linux KDC with Windows Built-in Kerberos Client and Vertica

A simple, but less common scenario is to configure Windows to authenticate against a non-Windows KDC. In this implementation, you use the `ksetup` utility to point the Windows operating system native Kerberos capabilities at a non-Active Directory KDC. By logging in to Windows, you obtain a ticket-granting ticket, similar to the Active Directory implementation. However, in this case, Windows is internally communicating with a Linux KDC. See the Microsoft Windows Server [Ksetup page](#) for more information.

Configure Windows Clients for Kerberos Authentication

Depending on which implementation you want to configure, refer to one of the following pages on the Microsoft Server website:

- To set up Windows clients with Active Directory, refer to [Step-by-Step Guide to Kerberos 5 \(krb5 1.0\) Interoperability](#).
- To set up Windows clients with the ksetup utility, refer to the [Ksetup page](#).

Authenticate and Connect Clients

The KDC can authenticate both an ADO.NET and a vsql client.

Note: Use the fully-qualified domain name as the server in your connection string; for example, use `host.example.com` instead of just `host`. That way, if the server moves location, you do not have to change your connection string.

Verify an ADO.NET Authentication Request and Connection

This example shows how to use the `IntegratedSecurity=true`, setting to specify that the ADO.NET driver authenticate the calling user's Windows credentials:

```
VerticaConnection conn = new  
VerticaConnection("Database=VMart;Server=host.example.com;  
Port=5433;IntegratedSecurity=true;  
KerberosServiceName=vertica;KerberosHostname=vcluster.example.com");  
conn.open();
```

Verify a vsql Authentication Request and Connection

1. Log in to your Windows client, for example, as `EXAMPLE\kuser`.
2. Run the vsql client and supply the connection string to Vertica:

```
C:\Users\kuser\Desktop>vsq1.exe -h host.example.com -K vcluster -U kuser  
  
Welcome to vsq1, the Vertica Analytic Database interactive terminal.  
Type: \h or \? for help with vsq1 commands  
\g or terminate with semicolon to execute query  
\q to quit
```

See Also

- [Kerberos Client/Server Requirements](#)
- [vsq1 Command Line Options](#) in Connecting to Vertica

- [ADO.NET Connection Properties](#) in Connecting to Vertica

Configure JDBC Clients on All Platforms

Kerberos authentication on JDBC clients uses Java Authentication and Authorization Service (JAAS) to acquire the initial Kerberos credentials. JAAS is an API framework that hides platform-specific authentication details and provides a consistent interface for other applications.

You specify the client login process through the JAAS Login Configuration File. This file contains options that specify the authentication method and other settings to use for Kerberos. A class called the `LoginModule` defines valid options in the configuration file.

The JDBC client principal is crafted as `jdbc-username@server-from-connection-string`.

Implement the LoginModule

Micro Focus recommends that you use the JAAS public class `com.sun.security.auth.module.Krb5LoginModule` provided in the Java Runtime Environment (JRE).

The `Krb5LoginModule` authenticates users using Kerberos protocols and is implemented differently on non-Windows and Windows platforms:

- **On non-Windows platforms:** The `Krb5LoginModule` defers to a native Kerberos client implementation. Thus, you can use the same `/etc/krb5.conf` setup as you use to [configure ODBC and vsqI clients](#) on Linux, HP-UX, AIX, MAC OSX, and Solaris platforms.
- **On Windows platforms:** The `Krb5LoginModule` uses a custom Kerberos client implementation bundled with the Java Runtime Environment (JRE). Windows settings are stored in a `%WINDIR%\krb5.ini` file, which has similar syntax and conventions to the non-Windows `krb5.conf` file. You can copy a `krb5.conf` from a non-Windows client to `%WINDIR%\krb5.ini`.

You can find documentation for the `LoginModules` in the `com.sun.security.auth` package, and on the [Krb5LoginModule](#) web page.

Create the JAAS Login Configuration

The [JAASConfigName connection property](#) identifies a specific configuration within a JAAS configuration that contains the `Krb5LoginModule` and its settings. The `JAASConfigName`

setting lets multiple JDBC applications with different Kerberos settings coexist on a single host. The default configuration name is `verticajdbc`.

Important: Carefully construct the JAAS login configuration file. If syntax is incorrect, authentication fails.

You can configure JAAS-related settings in the `java.security` master security properties file. This file resides in the `lib/security` directory of the JRE. For more information, see [Appendix A](#) in the Java™ Authentication and Authorization Service (JAAS) Reference Guide.

Create a JDBC Login Context

The following example shows how to create a login context for Kerberos authentication on a JDBC client. The client uses the default `JAASConfigName` of `verticajdbc` and specifies that:

- The ticket-granting ticket will be obtained from the ticket cache
- The user will not be prompted for a password if credentials cannot be obtained from the cache, keytab file, or through a shared state.

```
verticajdbc {  
    com.sun.security.auth.module.Krb5LoginModule  
    required  
    useTicketCache=true  
    doNotPrompt=true;  
};
```

JDBC Authentication Request and Connection

You can configure the `Krb5LoginModule` to use a cached ticket or keytab. The driver can also acquire a ticket or keytab automatically if the calling user provides a password.

In the preceding example, the login process uses a cached ticket and does not prompt for a password because both `useTicketCache` and `doNotPrompt` are set to `true`. If `doNotPrompt=false` and you provide a user name and password during the login process, the driver provides that information to the `LoginModule`. The driver then calls the `kinit` utility on your behalf.

1. On a JDBC client, call the `kinit` utility to acquire a ticket:

```
$ kinit kuser@EXAMPLE.COM
```

If you prefer to use a password instead of calling the `kinit` utility, see the next section.

2. Connect to Vertica:

```
Properties props = new Properties();
props.setProperty("user", "kuser");
props.setProperty("KerberosServiceName", "vertica");
props.setProperty("KerberosHostName", "vcluster.example.com");
props.setProperty("JAASConfigName", "verticajdbc");
Connection conn = DriverManager.getConnection
"jdbc:vertica://myserver.example.com:5433/VMart", props);
```

Have the Driver Acquire a Ticket

Sometimes, you may want to bypass calling the `kinit` utility yourself but still use encrypted, mutual authentication. In such cases, you can optionally pass the driver a clear text password to acquire the ticket from the KDC. The password is encrypted when sent across the network. For example, `useTicketCache` and `doNotPrompt` are both false in the following example. Thus, the calling user's credentials are not obtained through the ticket cache or keytab.

```
$ verticajdbc {
  com.sun.security.auth.module.Krb5LoginModule
  required
  useTicketCache=false
  doNotPrompt=false;
};
```

The preceding example demonstrates the flexibility of JAAS. The driver no longer looks for a cached ticket, and you do not have to call `kinit`. Instead, the driver takes the password and user name and calls `kinit` on your behalf.

See Also

- [Kerberos Client/Server Requirements](#)
- [JDBC Connection Properties](#) in Connecting to Vertica
- [Java™ Authentication and Authorization Service \(JAAS\) Reference Guide](#) (external website)

Troubleshooting Kerberos Authentication

These tips can help you avoid issues related to Kerberos authentication with Vertica Analytics Platform and to troubleshoot any problems that occur.

JDBC Client Authentication Fails

If Kerberos authentication fails on a JDBC client, check the JAAS login configuration file for syntax issues. If syntax is incorrect, authentication fails.

Working Domain Name Service (DNS) Not Configured

Verify that the DNS entries and the system host file (`/etc/hosts` or `/etc/hostnames`) on the network are all properly configured for your environment. If you are using a fully qualified domain name, ensure that is properly configured as well. Refer to the Kerberos documentation for your platform for details.

System Clocks Out of Sync

All Systems Except Red Hat 7/CentOS 7

System clocks in your network must remain in sync for Kerberos authentication to work properly. To do so:

1. Install NTP on the Kerberos server (KDC).
2. Install NTP on each server in your network.
3. Synchronize system clocks on all machines that participate in the Kerberos realm within a few minutes of the KDC and each other

Clock skew can be a problem on Linux virtual machines that need to sync with the Windows Time Service. Use the following steps to keep time in sync:

1. Using any text editor, open `/etc/ntp.conf`.
2. Under the `Undisciplined Local Clock` section, add the IP address for the Vertica Analytics Platform server. Then, remove existing server entries.
3. Log in to the server as root, and set up a cron job to sync time with the added IP address every half hour, or as often as needed. For example:

```
# 0 */2 * * * /etc/init.d/ntpd restart
```

4. Alternatively, run the following command to force clock sync immediately:

```
$ sudo /etc/init.d/ntpd restart
```

For more information, see [Set Up Time Synchronization](#) in Installing Vertica and the [Network Time Protocol website](#).

Red Hat 7/CentOS 7 Systems

In Red Hat 7/CentOS 7, ntpd is deprecated in favor of chrony. To keep system clocks in your network in sync for Kerberos authentication to work properly, do the following:

1. Install chrony on the Kerberos server (KDC).
2. Install chrony on each server in your network.
3. Synchronize system clocks on all machines that participate in the Kerberos realm within a few minutes of the KDC and each other.

Clock Skew on Linux Virtual Machines

Clock skew can be problematic on Linux virtual machines that need to sync with the Windows Time Service. Try the following to keep time in sync:

1. Using any text editor, open `/etc/chrony.conf`.
2. Under the `Undisciplined Local Clock` section, add the IP address for the Vertica Analytics Platform server. Then, remove existing server entries.
3. Log in to the server as root, and set up a cron job to sync time with the added IP address every half hour, or as often as needed. For example:

```
# 0 */2 * * * systemctl start chronyd
```

4. Alternatively, run the following command to force clock sync immediately:

```
$ sudo systemctl start chronyd
```

For more information, see [Set Up Time Synchronization](#) in Installing Vertica and the [Red Hat chrony guide](#).

Kerberos Ticket Is Valid, But Hadoop Access Fails

Vertica uses Kerberos tickets to obtain Hadoop tokens. It then uses the Hadoop tokens to access the Hadoop data. Hadoop tokens expire after a period of time, so Vertica periodically refreshes them. However, if your Hadoop cluster is set to expire tokens frequently, it is possible that tokens might not be refreshed in time. If the token expires, you cannot access data.

Setting the `HadoopFSTokenRefreshFrequency` configuration parameter allows you to specify how often Vertica should refresh the token. Specify this value, in seconds, to be smaller than the expiration period set for Hadoop. For example:

```
=> ALTER DATABASE exampledb SET HadoopFSTokenRefreshFrequency = '86400';
```

Encryption Algorithm Choices

Kerberos is based on symmetric encryption. Be sure that all Kerberos parties used in the Kerberos realm agree on the encryption algorithm to use. If they do not agree, authentication fails. You can review the exceptions in the `vertica.log`.

On a Windows client, be sure the encryption types match the types set on Active Directory. See [Configure Vertica for Kerberos Authentication](#).

Be aware that Kerberos is used only for securing the login process. After the login process completes, by default, information travels between client and server without encryption. If you want to encrypt traffic, use SSL. For details, see [Implementing SSL](#).

Kerberos Passwords Not Recognized

If you change your Kerberos password, you must re-create all of your keytab files.

Using the ODBC Data Source Configuration Utility

On Windows vsql clients, you may choose to use the ODBC Data Source Configuration utility and supply a client Data Source. If so, be sure you enter a Kerberos host name in the Client Settings tab to avoid client connection failures with the Vertica Analytic Database server.

Authentication Failure in Backup, Restore, or Admin Tools

This problem can arise in configurations where each Vertica node uses its own Kerberos principal. (This configuration is recommended.) When using vbr or admintools you might see an error such as the following:

```
$ vsql: GSSAPI continuation error: Miscellaenous failure
GSSAPI continuation error: Server not found in Kerberos database
```

Backup/restore and the admin tools use the value of KerberosHostname, if it is set, in the Kerberos principal used to authenticate. The same value is used on all nodes. If you have defined one Kerberos principal per node, as recommended, this value does not match. To correct this, unset the KerberosHostname parameter:

```
=> ALTER DATABASE mydb CLEAR KerberosHostname;
```

Server's Principal Name Does Not Match Host Name

This problem can arise in configurations where a single Kerberos principal is used for all nodes. Micro Focus recommends against using a single Kerberos principal for all nodes. Instead, use one principal per node and do not set the KerberosHostname parameter.

In some cases during client connection, the Vertica server's principal name might not match the host name in the connection string. (See also [Using the ODBC Data Source Configuration Utility](#) in this topic.)

On Windows vsql clients, you may choose to use the ODBC Data Source Configuration utility and supply a client Data Source. If so, be sure you enter a Kerberos host name in the Client Settings tab to avoid client connection failures with the Vertica Analytics Platform server.

On ODBC, JDBC, and ADO.NET clients, set the host name portion of the server's principal using the KerberosHostName connection string.

Tip: On vsql clients, you set the host name portion of the server's principal name using the `-K KRB HOST` command-line option. The default value is specified by the `-h` switch, which is the host name of the machine on which the Vertica server is running. `-K` is equivalent to the drivers' KerberosHostName connection string value.

For details, see [Command Line Options](#) in Connecting to Vertica.

Principal/Host Mismatch Issues and Resolutions

This section lists potential issues that may occur if the principal and host are mismatched.

- The `KerberosHostName` configuration parameter has been overridden.

For example, consider the following connection string:

```
jdbc:vertica://v_vmart_node0001.example.com/vmart?user=kuser
```

Because this connection string includes no explicit `KerberosHostName` parameter, the driver defaults to the host in the URL (`v_vmart_node0001.example.com`). If you overwrite the server-side `KerberosHostName` parameter as “abc”, the client generates an incorrect principal.

To resolve this issue, explicitly set the client’s `KerberosHostName` to the connection string, as in this example:

```
jdbc:vertica://v_vmart_node0001.example.com/vmart?user=kuser&kerberoshostname=abc
```

- Connection load balancing is enabled, but the node against which the client authenticates might not be the node in the connection string.

In this situation, consider changing all nodes to use the same `KerberosHostName` setting. When you use the default to the host that was originally specified in the connection string, load balancing cannot interfere with Kerberos authentication.

- You have a DNS name that does not match the Kerberos host name.

For example, imagine a cluster of six servers, where you want `hr-servers` and `finance-servers` to connect to different nodes on the Vertica Analytics Platform cluster. Kerberos authentication, however, occurs on a single (the same) KDC. In the following example, the Kerberos service host name of the servers is `server.example.com`.

Suppose you have the following list of example servers:

```
server1.example.com 192.16.10.11  
server2.example.com 192.16.10.12  
server3.example.com 192.16.10.13  
server4.example.com 192.16.10.14  
server5.example.com 192.16.10.15  
server6.example.com 192.16.10.16
```

Now, assume you have the following DNS entries:

```
finance-servers.example.com 192.168.10.11, 192.168.10.12, 192.168.10.13  
hr-servers.example.com 192.168.10.14, 192.168.10.15, 192.168.10.16
```

When you connect to `finance-servers.example.com`, specify:

- Kerberos `-h` host name option as `server.example.com`
- `-K` host option for `hr-servers.example.com`

For example:

```
$ vsql -h finance-servers.example.com -K server.example.com
```

- You do not have DNS set up on the client machine, so you must connect by IP only.

To resolve this issue, specify:

- Kerberos `-h` host name option for the IP address
- `-K` host option for `server.example.com`

For example:

```
$ vsql -h 192.168.1.12 -K server.example.com
```

- There is a load balancer involved (Virtual IP), but there is no DNS name for the VIP.

Specify:

- Kerberos `-h` host name option for the Virtual IP address
- `-K` host option for `server.example.com`

For example:

```
$ vsql -h <virtual IP> -K server.example.com
```

- You connect to Vertica using an IP address, but there is no host name to construct the Kerberos principal name.

Provide the instance or host name for the Vertica as described in [Inform About the Kerberos Principal](#)

- You set the server-side `KerberosHostName` configuration parameter to a name other than the Vertica node's host name, but the client cannot determine the host name based on the host name in the connection string alone.

Rename the KerberosHostName

Rename the KerberosHostName to match the name of the Vertica node's host name. For more information, see the following topics in Connecting to Vertica:

- [ODBC DSN Parameters](#)
- [JDBC Connection Properties](#)
- [ADO.NET Connection Properties](#)

LDAP Authentication

Lightweight Directory Access Protocol (LDAP) is an authentication method that works like password authentication. The main difference is that the LDAP method authenticates clients trying to access your Vertica database against an LDAP or Active Directory server. Use LDAP authentication when your database needs to authenticate a user with an LDAP or Active Directory server.

Before you configure LDAP authentication for your Vertica database, review the following information:

- [LDAP Prerequisites and Definitions](#)
- [LDAP Bind Methods](#)
- [Using LDAP Over SSL/TLS](#)
- [Workflow for Configuring LDAP Bind](#)
- [Workflow for Configuring LDAP Search and Bind](#)
- [Configuring Multiple LDAP Servers](#)

LDAP Prerequisites and Definitions

Prerequisites

Before you configure LDAP authentication for your Vertica database you must have:

- IP address and host name for the LDAP server. Vertica supports IPv4 and IPv6 addresses.
- Your organization's Active Directory information.
- A service account for search and bind.
- Administrative access to your Vertica database.
- `open-ldap-tools` package installed on at least one node. This package includes `ldapsearch`.

Definitions

The following definitions are important to remember for LDAP authentication:

Parameter name	Description
Host	IP address or host name of the LDAP server. Vertica supports IPv4 and IPv6 addresses. For more information, see IPv4 and IPv6 for Client Authentication .
Common name (CN)	Depending on your LDAP environment, this value can be either the username or the first and last name of the user.
Domain component (DC)	Comma-separated list that contains your organization's domain component broken up into separate values, for example: <pre>dc=vertica, dc=com</pre>
Distinguished name (DN)	<i>domain.com</i> . A DN consists of two DC components, as in "DC=example, DC= com".
Organizational unit (OU)	Unit in the organization with which the user is associated, for example, Vertica Users.
sAMAccountName	An Active Directory user account field. This value is usually the attribute to be searched when you use bind and search against the Microsoft Active Directory server.
UID	A commonly used LDAP account attribute used to store a username.
Bind	LDAP authentication method that allows basic binding using the DN.
Search and bind	LDAP authentication method that must log in to the LDAP server to

Parameter name	Description
	search on the specified attribute.
Service account	An LDAP user account that can be used to log in to the LDAP server during bind and search. This account's password is usually shared.
Anonymous binding	Allows a client to connect and search the directory (search and bind) without needing to log in.
ldapsearch	A command-line utility to search the LDAP directory. It returns information that you use to configure LDAP search and bind.
basedn	Distinguished name where the directory search should begin.
binddn	Domain name to find in the directory search.
search_attribute	Text to search for to locate the user record. The default is UID.

LDAP Parameters

There are several parameters that you need to configure for LDAP authentication.

General LDAP Parameters

Use the following parameters to configure for either LDAP bind or LDAP bind and search:

Parameter name	Description
host	LDAP server URI in the following format: <i>schema://host:optional_port</i> <i>schema</i> is either <code>ldap</code> (for LDAP/Active Directory) or <code>ldaps</code> (for secure LDAP/Active Directory).
starttls	Optional parameter that defines StartTLS behavior: <ul style="list-style-type: none">• <code>soft</code>—If the server does not support TLS, continue authenticating the user in plain text. This value is equivalent

Parameter name	Description
	<p>to the <code>-Z</code> option in <code>ldapsearch</code>.</p> <ul style="list-style-type: none"> <code>hard</code>—If server does not support TLS, authentication should fail. This value is equivalent to the <code>-ZZ</code> in <code>ldapsearch</code>. <p>Using <code>ldaps</code> is equivalent to <code>starttls='hard'</code>. However, if you use them together in the same connection string, authentication fails and the following error appears:</p> <pre>FATAL 3846: LDAP authentication failed for user "<user_name>"</pre>
<code>ldap_continue</code>	<p>When set to yes, this parameter allows a connection retry when a user not found error occurs during the previous connection attempt.</p> <p>For any other failure error, the system automatically retries the connection.</p>

LDAP Bind Parameters

Use the following parameters when authenticating with LDAP bind to create the bind name string. For more information see [Workflow for Configuring LDAP Bind](#).

Parameter name	Description
<code>binddn_prefix</code>	First half of the bind string.
<code>binddn_suffix</code>	<p>Second half of bind string.</p> <p>You must use the <code>binddn_prefix</code> and <code>binddn_suffix</code> together.</p> <p>In the following example, the bind name becomes <code>cn=<user_login_name>;ou=vertica users;dc=verticacorp;dc=com</code>.</p> <pre>=> ALTER AUTHENTICATION auth_method_name SET binddn_prefix='cn=',binddn_suffix=';ou=vertica users;dc=example;dc=com';</pre>
<code>domain_prefix</code>	<p>The domain where to find the user name.</p> <p>In the following example, the bind name is <code>verticacorp/<user_login_name></code></p>

Parameter name	Description
	<pre>ALTER AUTHENTICATION auth_method_name SET domain_prefix='Example';</pre>
email_suffix	The part of an email address that comes after the @ sign.

In the following example, the bind name becomes `<user_login_name>@example.com`.

```
=> ALTER AUTHENTICATION auth_method_name SET email_suffix='Example.com';
```

To create the bind name string, you must provide one of the following:

- Both `binddn_prefix` and `binddn_suffix`
- `domain_name`
- `email_suffix`

Otherwise, Vertica performs a bind and search operation instead of a bind operation.

LDAP Search and Bind Parameters

Use the following parameters when authenticating with LDAP search and bind. For more information see [Workflow for Configuring LDAP Search and Bind](#).

Parameter name	Description
basedn	Base DN for search.
binddn	Bind DN. Domain name to find in the directory search.
bind_password	Bind password. Required if you specify a binddn.
search_attribute	Optional attribute to search for on the LDAP server.

The following example shows how to set these three attributes. In this example, it sets

- `binddn` to `cn=Manager,dc=example,dc=com`
- `bind_password` to `secret`
- `search_attribute` to `cn`

```
=> ALTER AUTHENTICATION auth_method_name SET host='ldap://example13',  
basedn='dc=example,dc=com',binddn='cn=Manager,dc=example,dc=com',  
bind_password='secret',search_attribute='cn';
```

The `binddn` and `bind_password` parameters are optional. If you omit them, Vertica performs an anonymous search.

Using LDAP Over SSL/TLS

Vertica supports Transport Layer Security (TLS) for client authentication. TLS uses OpenSSL 0.9.8za and SSL v3/Transport Layer Security (TLS) 1.0 protocol.

The terms SSL and TLS are often used interchangeably. TLS is the successor to SSL and offers greater security. The original SSL standard was renamed TLS at the time it became open source. The introduction of TLS began with version 1, which is essentially equal to SSL 3. You use `openssl` commands to create certificates and keys and TLS syntax to create an authentication method.

For more information see the [Information Security website](#).

You use `ALTER AUTHENTICATION` to specify LDAP and SSL/TLS parameters. If you specify a host URL that starts with `ldaps`, the Vertica server authenticates using SSL/TLS on the specified port or on the secure LDAPS port (636).

```
ldaps://abc.dc.com
```

If the LDAP server does not support SSL on that port, authentication fails.

If you specify a host URL that starts with `ldap` and set the `LDAP_starttls` parameter, the Vertica server sends a StartTLS request. This request determines if the LDAP server supports TLS on the specified port or on the default LDAP port (389).

```
=> ALTER AUTHENTICATION ldap1 SET host='ldaps://abc.dc.com', binddn_prefix='CN=',  
binddn_suffix=',OU=Unit2,DC=dc,DC=com', basedn='dc=DC,dc=com',  
tls_cacert='/home/dc.com.ca.cer', tls_reqcert='never';
```

If the LDAP server does not support TLS on that port, the result depends on the value of the `starttls` parameter:

- `starttls = hard`: The Vertica server terminates the authentication process.
- `starttls = soft`: The Vertica server proceeds with the authentication but does not use TLS.

To configure LDAP over SSL/TLS, use the following configuration parameters:

Parameter Name	Description
TLS_REQCERT	<p><code>hard</code>—If the client does not provide a certificate, or provides an invalid certificate, it cannot connect. This is the default behavior.</p> <p><code>never</code>—The client does not request or verify a certificate.</p> <p><code>allow</code>—If the client does not provide a certificate or provides an invalid certificate, it can connect anyway.</p> <p><code>try</code>—If the client does not provide a certificate, they can connect. If the client provides an invalid certificate, they cannot connect.</p>
TLS_CADIR	<p>Path to the folder with the CA certificates. For example:</p> <pre>ALTER AUTHENTICATION ldap1 SET TLS_CADIR = '/scratch_b/qa/vertica/QA/VT_Scenario/V_SEC/';</pre>
TLS_CACERT	<p>Path to the CA certificate. For example:</p> <pre>ALTER AUTHENTICATION ldap1 SET TLS_CACERT = '/scratch_b/qa/vertica/QA/VT_Scenario/V_SEC/dc.com.ca.cer';</pre>

If you do not provide one or more of these parameters, the LDAP server checks to see if the `LDAPNOINIT` environment variable points to the `ldap.conf` file. If it does, the server uses the parameters specified in the `ldap.conf` file. If the LDAP server cannot find the `ldap.conf` file, authentication fails.

The following example shows how to specify the TLS parameters and the LDAP parameters when configuring LDAP over SSL/TLS:

```
=> CREATE AUTHENTICATION LDAP1 METHOD 'ldap' HOST :clientIP = '172.16.65.177';
=> GRANT AUTHENTICATION ldap1 TO user1;
=> ALTER AUTHENTICATION ldap1 SET host='ldaps://abc.dc.com', binddn_prefix='CN=',
binddn_suffix=',OU=Unit2,DC=dc,DC=com', basedn='dc=DC,dc=com',
tls_cacert='/home/dc.com.ca.cer', tls_reqcert='never';
```

Configuring Multiple LDAP Servers

If you need to configure multiple LDAP servers that have different URLs, create a separate authentication record for each server. Use the `PRIORITY` keyword to indicate which search the LDAP server performs first.

The following statements create two authentication methods, `vldap1` and `vldap2`. They specify that the LDAP server first search the entire directory (`basedn=dc=example,dc=com`) for a DN with an OU attribute `Sales`. If the first search returns no results, or otherwise fails, the LDAP server next searches for a DN with the OU attribute `Marketing`:

```
=> CREATE AUTHENTICATION vldap1 method "ldap" HOST 10.0.0.0/8;
=> ALTER AUTHENTICATION vldap1 SET
    host='ldap://ldap.example.com/search',
    basedn='dc=example,dc=com',
    search_attribute='Sales'
    PRIORITY 1;
=> GRANT AUTHENTICATION vldap1 to public;

=> CREATE AUTHENTICATION vldap2 method "ldap" HOST 10.0.0.0/8;
=> ALTER AUTHENTICATION vldap2 SET
    host='ldap://ldap.example.com/search',
    basedn='dc=example,dc=com',
    search_attribute='Marketing'
    PRIORITY 0;
=> GRANT AUTHENTICATION vldap2 to public;
```

LDAP Bind Methods

There are two LDAP methods that you use to authenticate your Vertica database against an LDAP server.

- **Bind**—Use LDAP bind when Vertica connects to the LDAP server and binds using the CN and password. (These values are the username and password of the user logging into the database). Use the bind method when your LDAP account's CN field matches that of the username defined in your database. For more information see [Workflow for Configuring LDAP Bind](#).
- **Search and Bind** —Use LDAP search and bind when your LDAP account's CN field is a user's full name or does not match the username defined in your database. For search and bind, the username is usually in another field such as UID or sAMAccountName in a standard Active Directory environment. Search and bind requires your organization's Active Directory information. This information allows Vertica to log into the LDAP server and search for the specified field. For more information see [Workflow for Configuring LDAP Search and Bind](#).

If you are using search and bind, having a service account simplifies your server side configuration. In addition, you do not need to store your Active Directory password.

LDAP Anonymous Binding

Anonymous binding is an LDAP server function. Anonymous binding allows a client to connect and search the directory (bind and search) without logging in because `binddn` and `bindpasswd` are not needed.

You also do not need to log in when you configure LDAP authentication using Management Console.

Workflow for Configuring LDAP Bind

To configure your Vertica database to authenticate clients using LDAP bind, follow these steps:

1. Obtain a service account, as described in [LDAP Configuration Considerations](#). You cannot use the service account in the connection parameters for LDAP bind.
2. Compare the user's LDAP account name to their Vertica username. For example, if John Smith's Active Directory (AD) `sAMAccountName` = `jsmith`, his Vertica username must also be `jsmith`.

However, the LDAP account does not have to match the database user name, as shown in the following example:

```
=> CREATE USER r1 IDENTIFIED BY 'password';
=> CREATE AUTHENTICATION ldap1 METHOD 'ldap' HOST '172.16.65.177';
=> ALTER AUTHENTICATION ldap1 SET HOST=
'ldap://172.16.65.10',basedn='dc=dc,dc=com',binddn_suffix=',ou=unit2,dc=dc,dc=com',binddn_
prefix='cn=use';
=> GRANT AUTHENTICATION ldap1 TO r1;

\! ${TARGET}/bin/vsql -p $PGPORT -U r1 -w $LDAP_USER_PASSWD -h ${HOSTNAME} -c
"select user_name, client_authentication_name from sessions;"
 user_name | client_authentication_name
-----+-----
 r1        | ldap
(1 row)
```

3. Run `ldapsearch` from a Vertica node against your LDAP or AD server. Verify the connection to the server and identify the values of relevant fields. Running `ldapsearch` helps you build the client authentication string needed to configure LDAP authentication.

In the following example, `ldapsearch` returns the CN, DN, and `sAMAccountName` fields (if they exist) for any user whose CN contains the username `jsmith`. This search succeeds only for LDAP servers that allow anonymous binding:

```
$ ldapsearch -x -h 10.10.10.10 -b "ou=Vertica Users,dc=CompanyCorp,dc=com"
'(cn=jsmith*)' cn dn uid sAMAccountName
```

`ldapsearch` returns the following results. The relevant information for LDAP bind is in **bold**:

```
# extended LDIF
#
# LDAPv3
# base <ou=Vertica Users,dc=CompanyCorp,dc=com> with scope subtree
# filter: (cn=jsmith*)
# requesting: cn dn uid sAMAccountName
#
# jsmith, Users, CompanyCorp.com
dn:cn=jsmith,ou=Vertica Users,dc=CompanyCorp,dc=com
cn: jsmith
uid: jsmith
# search result
search: 2
result: 0 Success
# numResponses: 2
# numEntries: 1
```

4. Create a new authentication record based on the information from `ldapsearch`. In the `ldapsearch` entry, the CN is username `jsmith`, so you do not need to set it. Vertica automatically sets the CN to the username of the user who is trying to connect. Vertica uses that CN to bind against the LDAP server.

```
=> CREATE AUTHENTICATION v_ldap_bind METHOD 'ldap' HOST '0.0.0.0/0';
=> GRANT AUTHENTICATION v_ldap_bind TO public;
=> ALTER AUTHENTICATION v_ldap_bind SET
host='ldap://10.10.10.10/',
basedn='DC=CompanyCorp,DC=com',
binddn_prefix='cn=',
binddn_suffix=',OU=Vertica Users,DC=CompanyCorp,DC=com';
```

For more information see [LDAP Bind Parameters](#)

Workflow for Configuring LDAP Search and Bind

To configure your Vertica database to authenticate clients using LDAP search and bind, follow these steps:

1. Obtain a service account, as described in [LDAP Configuration Considerations](#).
2. From a Vertica node, run `ldapsearch` against your LDAP or AD server. Verify the connection to the server, and identify the values of relevant fields. Running `ldapsearch` helps you build the client authentication string needed to configure LDAP authentication.

In the following example, `ldapsearch` returns the CN, DN, and `sAMAccountName` fields (if they exist) for any user whose CN contains the username, John. This search succeeds only for LDAP servers that allow anonymous binding:

```
$ ldapsearch -x -h 10.10.10.10 -b 'OU=Vertica Users,DC=CompanyCorp,DC=com' -s sub -D 'CompanyCorp\jsmith' -W '(cn=John*)' cn dn uid sAMAccountName
```

3. Review the results that `ldapsearch` returns. The relevant information for search and bind is in bold:

```
# extended LDIF
#
# LDAPv3
# base <OU=Vertica Users,DC=CompanyCorp,DC=com> with scope subtree
# filter: (cn=John*)
# requesting: cn dn sAMAccountName
#
# John Smith, Vertica Users, CompanyCorp.com
dn: CN=jsmith,OU=Vertica Users,DC=CompanyCorp,DC=com
cn: Jsmith
sAMAccountName: jsmith
# search result
search: 2
result: 0 Success
# numResponses: 2
# numEntries: 1
```

4. Create the client authentication record. The `cn` attribute contains the username you want—`jsmith`. Set your search attribute to the `CN` field so that the search finds the appropriate account.

```
=> CREATE AUTHENTICATION v_ldap_bind_search METHOD 'ldap' HOST '10.10.10.10';
=> GRANT AUTHENTICATION v_ldap_bind_search TO public;
=> ALTER AUTHENTICATION v_ldap_bind_search SET
host='ldap://10.10.10.10',
```

```
basedn='OU=Vertica,DC=CompanyCorp,DC=com',  
binddn='CN=jsmith,OU=Vertica Users,DC=CompanyCorp,DC=com',  
bind_password='password',  
search_attribute='CN';
```

For more information see [LDAP Search and Bind Parameters](#)

TLS/SSL Server Authentication

The terms SSL and TLS are often used interchangeably. This document uses both terms. *TLS* is the successor to SSL and offers greater security. The original *SSL* standard was renamed TLS at the time it became open source. The introduction of TLS began with version 1, which is essentially equal to SSL 3. You use `openssl` commands to create certificates and keys and TLS syntax to create an authentication method.

SSL Authentication

To protect privacy and verify data integrity, you can configure Vertica and database clients to use Secure Socket Layer (SSL). SSL allows secure connection and communication between the client and the server. The SSL protocol uses a trusted third party called a *Certificate Authority (CA)*. Both the owner of a certificate and the party that relies on the certificate trust the CA.

Vertica supports the following authentication methods under Transport Layer Security (TLS) v1.0, v1.1, and v1.2 protocol:

- **SSL server authentication** — Lets the client confirm the server's identity. The client verifies that the server's certificate and public key are valid and were issued by a certificate authority (CA) listed in the client's list of trusted CAs. This authentication helps prevent man-in-the-middle attacks. See "Prerequisites for SSL Server Authentication and SSL Encryption" in [SSL Overview](#) and [Configuring SSL](#).
- **SSL client authentication** — (Optional) Lets the server confirm the client's identity. The server verifies that the client's certificate and public key are valid and were issued by a certificate authority (CA) listed in the server's list of trusted CAs. Client authentication is optional because Vertica can authenticate the client at the application protocol level through user name and password credentials. See "Optional Prerequisites for SSL Server and Client Mutual Authentication" in [SSL Overview](#).
- **Encryption** — Encrypts data sent between the client and database server. This method significantly reduces the likelihood that the data can be read if the connection between the client and server is compromised. Encryption works at both ends of a transaction, regardless of whether SSL Client Authentication is enabled. See "Prerequisites for SSL Server Authentication and SSL encryption" in [SSL Overview](#) and [Configuring SSL](#).
- **Data integrity** — Verifies that data sent between the client and server has not been altered during transmission.

Note: For server authentication, Vertica supports using RSA encryption with [ephemeral Diffie-Hellman](#) (DH). DH is the key agreement protocol.

Certificate Authority

The CA issues electronic certificates to identify one or both ends of a transaction. These certificates to verify ownership of a public key by the name on the certificate.

Public and Private Keys

A CA issues digital certificates that contain a public key and the identity of the owner.

The public key is available to all users through a publicly accessible directory. However, private keys are confidential to their respective owners. When you use a private/public key pair, the data is encrypted by one key and decrypted by its corresponding key.

- If encrypted with a public key, data can be decrypted by its corresponding private key only.
- If encrypted with a private key, data can be decrypted by its corresponding public key only.

For example, suppose Alice wants to send confidential data to Bob. Because she wants only Bob to read it, she encrypts the data with Bob's public key. Even if someone else gains access to the encrypted data, it remains protected. Because only Bob has access to his corresponding private key, he is the only person who can decrypt Alice's encrypted data back into its original form.

SSL Overview

Before you implement SSL security, including mutual mode, obtain the appropriate certificate signed by a certificate authority (CA) and private key files. Copy the certificate file to the database catalog directory. These files must use the Privacy-Enhanced Mail (PEM) format. PEM is the standard file format for Certificates and can be included in ascii or rich text documents. For reference information on SSL use the following links:

- [OpenSSL Documentation](#)
- [OpenSSL Support](#)

- [OpenSSL FAQs](#)
- [OpenSSL Release](#)

Use the following files for SSL authentication:

- `root.crt` - contains the top-level Certificate Authorities that are trusted for signing server certificates (`server.crt`).
- `server.crt` - must reside in the server's data directory and contains the trusted server certificate. This file gets sent to the client where `root.crt` identifies the server.
- `server.key` - must reside in the server's data directory and proves the server certificate was sent by the certificate owner. It does not indicate the certificate owner is trustworthy.

If you make changes to any of these files you must restart the server.

You can also implement SSL with LDAP authentication. For more information see [Using LDAP Over SSL/TLS](#).

Using wildcards

You can enter wildcard characters as part of the server names in `root.crt`. For example, `server.crt` contains server names called `eng001.corptech.com`, `eng002.corptech.com`, and `eng003.corptech.com`. You can enter `*.corptech.com` in `root.crt` and it locates the required server.

The wildcard must be the first character of the hostname followed by a period, for example:

```
*.hostname.com
```

Set Up SSL Server Authentication and SSL Encryption

Follow these steps to set up server SSL authentication:

Important: If you do not perform these steps, database operation may be compromised. If the client cannot authenticate the server, the database does not start.

1. Enable SSL authentication in one of the following ways:

Set `EnableSSL=1` in `vertica.conf`

Enter `ALTER DATABASE mydb SET EnableSSL = 1;` in `vsq`.

2. Copy the server certificate file (`server.crt`) and private key (`server.key`) to one of your server hosts in the cluster, as follows:
3. Distribute these files to all server hosts using the instruction in [Distributing Certificates and Keys](#).

The public key contained in the certificate and the corresponding private key allow the SSL connection to encrypt the data to protect data integrity

4. SSL Server mode requires that the client verify the server's certificate. The client must be able to access certificate authority file (`root.crt`), and the server must be set with server certificate and private key. Also, SSL Mutual mode requires that the server verify the client's certificate. The server should be set with CA file and the client must have the client certificate and its private key.

For vsql:

If the `VSQL_HOME` environment variable is not set, copy the `root.crt` file to the `.vsq1` subdir of the login user's home directory (for example, `~/.vsq1/root.crt`).

If the `VSQL_HOME` environment variable is set, copy the `root.crt` file to the `.vsq1` subdir of the target directory (for example, `$vsq1_home/.vsq1/root.crt`)

The `root.crt` file contains the Certificate Authority that issued the server certificate.

Set SSL Server for Mutual Mode Authentication

Use SSL Mutual Mode to have both server and client mutually authenticate themselves with SSL keys. With SSL Mutual Mode the server requests a certificate from the client and the client requests a certificate from the server. Set up SSL Mutual Mode as follows:

1. Enable SSL authentication in one of the following ways:

Set `EnableSSL=1` in `vertica.conf`

Enter `ALTER DATABASE mydb SET EnableSSL = 1;` in `vsq1`.

2. Copy the `root.crt` file to one server host in the cluster. This file is distributed to all server hosts when you distribute certificates and keys. See [Distributing Certificates and Keys](#).

The `root.crt` file has the same name on the client and server though the file contents can differ. The contents are identical only if the client and server certificates were used by the same root certificate authority (CA).

3. Copy the client certificate file (`client.crt`) and private key (`client.key`) to each client. For `vsq!`:

- If the `VSQ!_HOME` environment variable is set, copy the file to the `.vsq!` subdirectory of the target directory set up in the environment variable (e.g., `$vsq!_home/.vsq!/client.crt`).
- If the `VSQ!_HOME` environment variable is not set, copy the two files to the `.vsq!` subdirectory of the login user's home directory. (e.g., `~/.vsq!/client.crt`).

If you are using either ODBC or JDBC, you can place the files anywhere on your system. Then, provide the location in the connection string (ODBC/JDBC) or ODBCINI (ODBC only). See [Configuring SSL for ODBC Clients](#) and [Configuring SSL for JDBC Clients](#).

Important: If you're using ODBC, the private key file (`client.key`) must have read and write permissions only for the `dbadmin` user. For example:

```
chmod 600 client.key
```

Do not provide any additional permissions or extend them to any other users.

Generating SSL Certificates and Keys

Generating SSL certificates and keys, you must perform the following tasks:

- [Create a Certificate Authority Private Key and Public Certificate](#) that can then be used to sign server and client keys.

Important: In a production environment, always use certificates signed by a Certificate Authority.

- [Create the Server Private key and Certificate](#), and request a new server certificate that includes a public key.

- [Create the Client Private key and Certification](#), and request a new client certificate that includes a public key.

For more detailed information on creating signed certificates, refer to the [OpenSSL documentation](#).

The documentation includes examples and sample procedures to show how to create certificates and keys. The commands shown allow many other possible options not used in these examples. Create commands based on your specific environment.

Create a Certificate Authority Private Key and Public Certificate

Create a Certificate Authority (CA) private key and public certificate. For more information on using the following commands, see the [OpenSSL documentation](#).

1. Generate CA files `serverca.crt` and `servercakey.pem`. This allows the signing of server and client keys:

```
$ openssl genrsa -out new_servercakey.pem
$ openssl req -config openssl_req_CA.conf -new -x509 -key servercakey.pem -out
serverca.crt
```

You can add multiple CA files to the `.crt` file with the following command:

```
$ cat serverca_new.crt >> serverca.crt
```

This concatenates the `serverca_new.crt` file to the original CA file `serverca.crt`. You can run this command multiple times to concatenate additional CA files.

2. Create the server private key (`server.key`) and public key (`server.crt`):

```
$ openssl genrsa -out server.key
$ openssl req -config openssl_req_server.conf -new -key server.key -out server_reqout.txt
$ openssl x509 -req -in server_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt
-CAkey servercakey.pem -out server.crt
```

3. Create the client private key (`client.crt`) and public key (`client.key`):

```
$ openssl genrsa -out client.key
$ openssl req -config openssl_req_client.conf -new -key client.key -out client_reqout.txt
$ openssl x509 -req -in client_reqout.txt -days 3650 -sha1 -CAcreateserial
-CA serverca.crt -CAkey servercakey.pem -out client.crt
```

4. Enter the following sample CA certificate values in response to `openssl` command line prompts. The actual values you enter here will be different than the sample values.

Rather than enter these values from command line prompts, you can optionally provide the same information in `.conf` files. For example, `openssl_req_ca.conf` in the preceding example.

```
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (e.g., city) [Newbury]:Cambridge
Organization Name (e.g., company) [My Company Ltd]:CorpName
Organizational Unit Name (e.g., section) []:TechSupport
Common Name (e.g., your name or server hostname) []:myhost
Email Address []:myhost@CorpName.com
```

5. Include one unique Distinguished Name (DN) for each certificate that you create. In the preceding examples, the DN is the Organizational Unit Name.
6. Set file permissions:

```
$ chmod 700 server.crt server.key
$ chmod 700 client.crt client.key
```

7. Rename the CA file `serverca.crt` to `root.crt`, and do one of the following:

- copy it to `VSQL_HOME` or
- point to it in [ODBC DSN Configuration dialog](#)

8. Create a trusted Certificate Authority for the client for server mutual mode:

```
=> ALTER DATABASE <mydb> SET SSLCA = '<content of serverca.crt>';
```

You now have a CA private key, `new_servercakey.pem`. You also have a CA public certificate, `new_serverca.crt`. Use both the private key and the public certificate in the procedures that follow for creating server and client certificates.

Generating Certificates and Keys for MC

A *certificate signing request (CSR)* is a block of encrypted text generated on the server on which the certificate is used. You send the CSR to a certificate authority (CA) to apply for a digital identity certificate. The CA uses the CSR to create your SSL certificate from information in your certificate; for example, organization name, common (domain) name, city, and country.

Management Console (MC) uses a combination of OAuth (Open Authorization), Secure Socket Layer (SSL), and locally-encrypted passwords to secure HTTPS requests between a user's browser and MC, and between MC and the agents. Authentication occurs through MC and between agents within the cluster. Agents also authenticate and authorize jobs.

The MC configuration process sets up SSL automatically, but you must have the `openssl` package installed on your Linux environment first.

When you [connect to MC](#) through a client browser, Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your Vertica cluster, and on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See [Set Up Time Synchronization](#) in [Installing Vertica](#).

Create a Certificate and Submit it for Signing

For production, you must use certificates signed by a certificate authority. You can create and submit a certificate and when the certificate returns from the CA, [import the certificate into MC](#).

Use the `openssl` command to generate a new CSR:

```
$ sudo openssl req -new -key /opt/vconsole/config/keystore.key -out server.csr
```

When you press **Enter**, you are prompted to enter information to be incorporated into your certificate request. Some fields contain a default value, which you should change for security reasons. Other fields you can leave blank, such as password and optional company name. To leave the field blank, type ' '.

Important: The `keystore.key` value for the `-key` option creates private key for the keystore. If you generate a new key and import it using the Management Console interface, the MC process does restart properly. You must restore the original `keystore.jks` file and [restart Management Console](#).

This information is contained in the CSR and shows both the default and replacement values:

```
Country Name (2 letter code) [GB]:USState or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (eg, city) [Newbury]: Cambridge
Organization Name (eg, company) [My Company Ltd]:Micro Focus
Organizational Unit Name (eg, section) []:Information Management
Common Name (eg, your name or your server's hostname) []:console.vertica.com
Email Address []:mcadmin@vertica.com
```

The **Common Name** field is the fully qualified domain name of your server. Your entry must exactly match what you type in your web browser, or you receive a name mismatch error.

Self-Sign a Certificate for Testing

To test your new SSL implementation, you can self-sign a CSR using either a temporary certificate or your own internal CA, if one is available.

Note: A self-signed certificate generates a browser-based error notifying you that the signing certificate authority is unknown and not trusted. For testing purposes, accept the risks and continue.

The following command generates a temporary certificate, which expires after 365 days:

```
$ sudo openssl x509 -req -days 365 -in server.csr -signkey /opt/vconsole/config/keystore.key -out server.crt
Enter passphrase for /opt/vconsole/config/keystore.key:
Enter same passphrase again:
```

The previous example prompts you for a passphrase. This is required for Apache to start. To implement a passphrase you must put the `SSLPassPhraseDialog` directive in the appropriate Apache configuration file. For more information see your Apache documentation.

This example shows the command's output to the terminal window:

```
Signature oksubject=/C=US/ST=Massachusetts/L=Billerica/O=Micro Focus/OU=IT/
CN=console.vertica.com/emailAddress=mcadmin@vertica.com
Getting Private key
```

You can now [import the self-signed key](#), `server.crt`, into Management Console.

See Also

- [Configuring SSL](#)
- [Key and Certificate Management Tool](#)

Importing a New Certificate to MC

Use this procedure to import a new certificate into Management Console.

Note: To generate a new certificate for Management Console, you must use the `keystore.key` file, which is located in `/opt/vconsole/config` on the server on which

you installed MC. Any other generated key/certificate pair causes MC to restart incorrectly. You will then have to restore the original keystore .jks file and [restart Management Console](#). See [Generating Certifications and Keys for Management Console](#).

1. [Connect to Management Console](#), and log in as an administrator.
2. On the Home page, click **MC Settings**.
3. In the button panel on the left, click **SSL certificates**.
4. To the right of "Upload a new SSL certificate," click **Browse** to import the new key.
5. Click **Apply**.
6. [Restart Management Console](#).

Configuring SSL

Configure SSL for each server in the cluster.

1. Verify that you have performed at least the minimum steps required in [SSL Overview](#) for server authentication and encryption and, optionally, for mutual authentication.
2. Verify that you have performed the steps in [Distributing Certificates and Keys](#).

Important: Before you set the [Security Parameters](#) SSLCertificate and SSLPrivateKey, you must first set the EnableSSL parameter. Admintools sets these parameters for you when you perform the procedure steps listed in [Distributing Certificates and Keys](#). Alternatively, you can use vsql to set the parameters using the ALTER DATABASE statement. For more information on setting configuration parameters see [ALTER DATABASE](#).

These parameters are also automatically set during upgrade to 7.1 if you set EnableSSL=1 in the previous version.

3. Set the EnableSSL parameter to *1*. By default, EnableSSL is set to 0 (disabled).

```
=> ALTER DATABASE mydb SET EnableSSL = 1;
```

4. [Restart the database](#).
5. If you are using either ODBC or JDBC, configure SSL for the appropriate client:

- [Configuring SSL for ODBC Clients](#)
- [Configuring SSL for JDBC Clients](#)

vsqI automatically tries to connect using SSL. If a connection fails, and your server is started in SSL Server Mode, vsqI attempts to make a second connection over clear text. If you start the server in SSL Mutual Mode, the connection fails without vsqI attempting to connect over clear text.

Configure JDBC for SSL Mutual Mode

In addition to the procedure above, you need to perform the following to configure SSL in Mutual Mode:

- a. Edit `openssl_req_server.conf` as follows:

```
[req]
prompt                = no
distinguished_name    = CStore4Ever
req_extensions        = v3_req

[CStore4Ever]
C                    = US
ST                  = Massachusetts
O                   = Corp Server
CN                  = engXXX
emailAddress        = foo@bar.com

[v3_req]
basicConstraints     = CA:FALSE
keyUsage             = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName       = @alt_names

[alt_names]
DNS.1 = engXXX.corp.com
DNS.2 = engXXX
DNS.3 = *.corp.com
IP = 10.20.40.XX
```

- b. Create the server private key (`server.crt`) and public key (`server.key`):

```
$ openssl genrsa -out server.key
$ openssl req -config openssl_req_server.conf -new -key server.key -out server_reqout.txt
$ openssl x509 -req -in server_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt
  -CAkey servercakey.pem -extensions v3_req -extfile openssl_req_server.conf -out server.crt
```

Note: If you are using SSL mutual mode with JDBC, copy the `root.crt` file to a location on any one of the clients. After you copy the file to the client, `root.crt` is

incorporated into the truststore. For more information, see [Generating SSL Certificates and Keys](#).

Configuring SSL for ODBC Clients

Configuring SSL for ODBC clients requires that you set the SSLMode connection property. If you want to configure optional SSL client authentication, you must also configure the [Security Parameters](#) SSLKeyFile and SSLCertFile connection properties.

How you configure the DSN depends on your operating system:

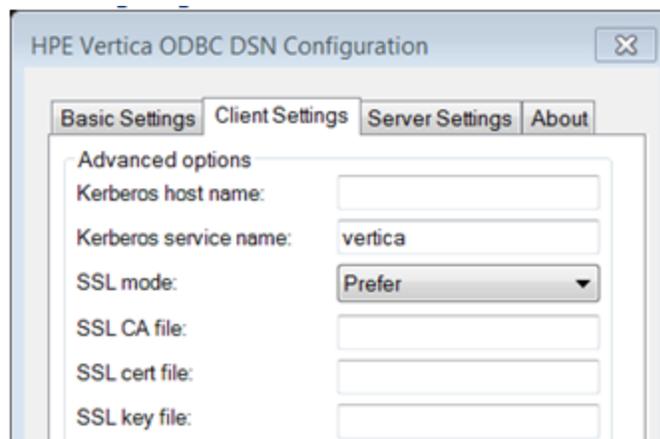
- Linux and UNIX — Enter the connection properties in the `odbc.ini` file. See [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX Clients](#).
- Microsoft Windows — Enter the connection properties in the Windows Registry. See [Creating an ODBC DSN for Windows Clients](#).

For Windows ODBC you can set connection string properties in the Client Settings tab on the ODBC DSN Configuration dialog. Connection string properties you can set are:

SSL CA file - the location of the root certificate file, for example `root.crt`.

SSL cert file - the location of the server certificate file, for example `server.crt`.

SSL key file - the location of the server key file, for example, `server.key`.



Set SSLMode Connection Property

Set the SSLMode connection property to one of the following options for the DSN:

Property	Description
verify_full	Encrypts data and connects to a user-specified trusted server.
verify_ca	Encrypts data and connects to a trusted server.
require	Requires the server to use SSL. If the server cannot provide an encrypted channel, the connection fails.
prefer	(Default value) Indicates your preference that the server to use SSL. The first connection to the database tries to use SSL. If that connection fails, a second connection is attempted over a clear channel.
allow	Makes a connection to the server whether the server uses SSL or not. The first connection attempt to the database is made over a clear channel. If that connection fails, a second connection is attempted over SSL.
disable	Never connects to the server using SSL. This setting is typically used for troubleshooting.

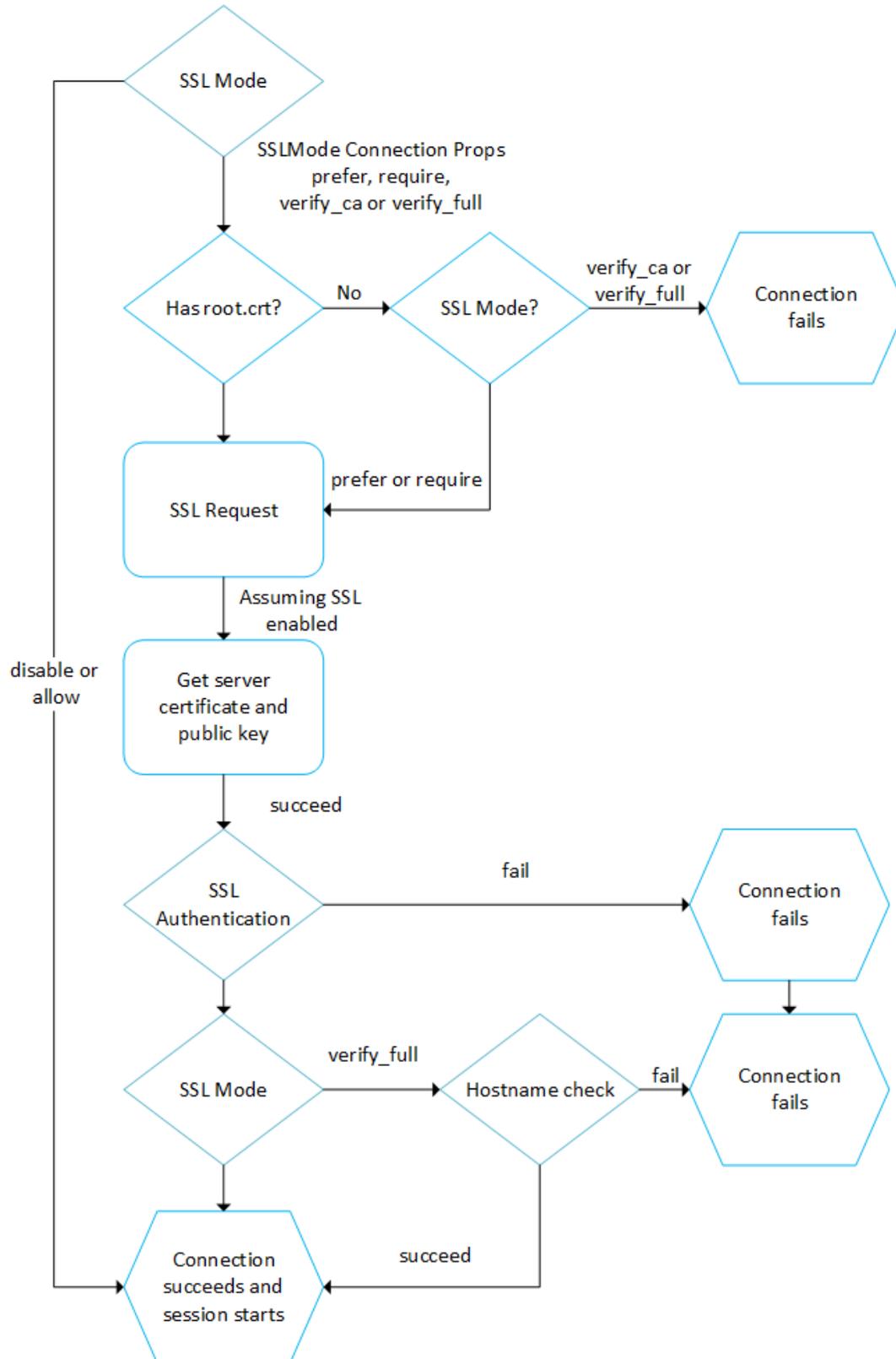
Using verify_ca and verify_full

You can use the SSL Mode properties `verify_ca` and `verify_full` exclusively on client authentication. These properties require:

- The `root.crt` to allow the client to access the server's authentication certificate. If `root.crt` does not exist, and `verify_ca` or `verify_full` are set, the connection fails upon SSL initialization.
- A `server.crt`, which contains the authentication certificate.
- A `server.key`, which contains the server's private key that prevents the server from accessing external systems.
- A server hostname that matches the hostname specified by the client, `verify_full` only.
- A certificateAll nodes in the cluster must either share the same certificate or each certificate must contain all the hostnames or IP addresses in the Subject Alternative Name (SAN).

SSL Workflow

The following diagram shows an example workflow for SSL authentication. Your actual workflow may differ depending on what SSLMode Connection Properties you use.



In this workflow:

- If the SSLMode Connection Property is set to none or allow, the client connects without authentication.
- If the SSLMode Connection Property is set to verify_ca or verify_full, and root.crt does not exist, the SSL authentication fails. If root.crt exists, the authentication process proceeds.
- During SSL authentication if the SSLMode Connection Property is set to verify_full, and the server hostname is not the same as the hostname specified by the client, authentication fails.

Enable SSL Mutual Mode Authentication

You can optionally configure SSL mutual mode by setting the following database [Security Parameters](#):

- SSLKeyFile — Set this connection property to the file path and name of the client's private key. This key can reside anywhere on the client.
- SSLCertFile — Set this connection property to the file path and name of the client's public certificate. This file can reside anywhere on the client.

Configuring SSL for JDBC Clients

Configuring SSL Authentication for JDBC clients involves involves the following tasks:

- Set required properties
- Optionally run the SSL Debug Utility
- Configure for SSL Mutual Mode (optional)

Set Required Properties

Set Properties When Location or the Keystore/Truststore Is Not the Default

If you are using a location or the keystore/truststore that is not the default, set the following system properties so that the JRE can find your keystore/truststore:

```
$ javax.net.ssl.keyStore  
$ javax.net.ssl.trustStore
```

Set Properties When Keystore/Truststore Is Password Protected

If your keystore/truststore is password protected, set the following system properties so that the JRE has access to it:

```
$ javax.net.ssl.keyStorePassword  
$ javax.net.ssl.trustStorePassword
```

Run the SSL Debug Utility

After configuring SSL for JDBC, optionally run the following command to enable the debug utility for SSL:

```
$ -Djavax.net.debug=ssl
```

You can use several debug specifiers (options) with the debug utility. The specifiers help narrow the scope of the debugging information that is returned. For example, you could specify one of the options that prints handshake messages or session activity.

For information on the debug utility and its options, see Debugging Utilities in the Oracle document, [JSSE Reference Guide](#).

For information on interpreting debug information, refer to the Oracle document, [Debugging SSL/TLS Connections](#).

Distributing Certificates and Keys

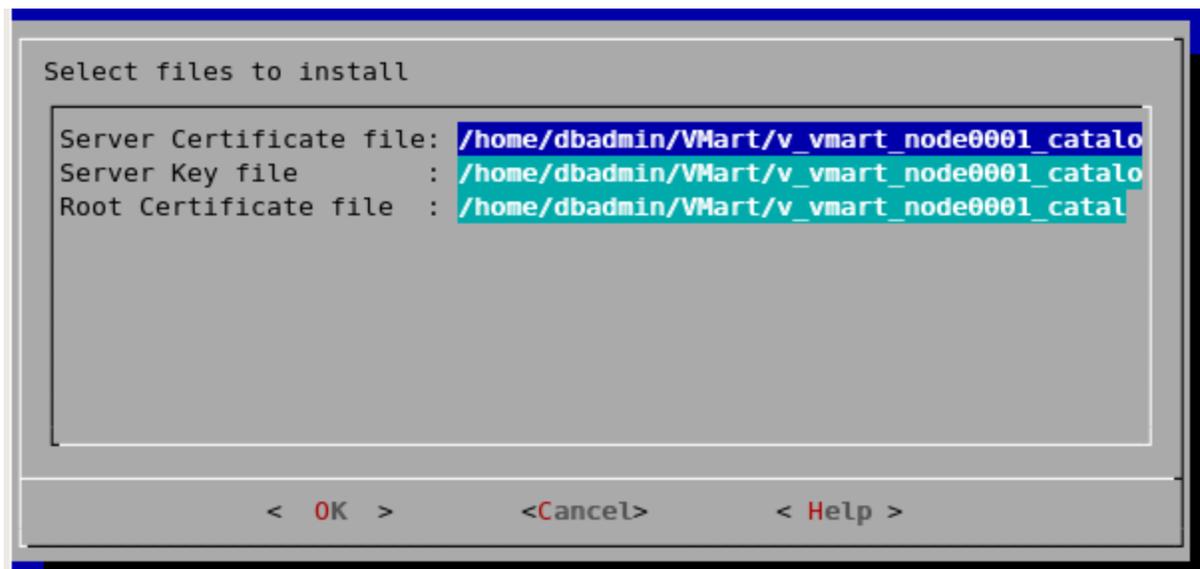
Before you can distribute certifications and keys to all hosts in a cluster, you must obtain the appropriate certificate signed by a certificate authority (CA) and private key files. See [SSL Overview](#).

To distribute certifications and keys to all hosts in a cluster:

1. Log on to a host that contains the certifications and keys you want to distribute.
2. Start the Administration Tools, as described in [Using the Administration Tools](#)

Note: The database does not need to be running when you distribute the certificates and key files.

3. On the **Main Menu** in the Administration Tools, select **Configuration Menu**, and click **OK**.
4. On the **Configuration Menu**:
 - a. Select **Distribute Config Files**, and click **OK**.
 - b. Select **SSL Keys**, and click **OK**.
5. Select the database on which you want to distribute the files and click **OK**. The following appears:



6. [Optional] Modify the fields in the previous screenshot to add the file locations for the `server.crt`, `server.key` and `root.crt` files, and click **OK** to distribute the files.

Admintools sets the parameters `SSLCertificate`, `SSLPrivateKey`, and, if applicable, `SSLCA`. See [Security Parameters](#).

- If you are upgrading to 7.1, the `SSLCertificate` and `SSLPrivateKey` parameters are automatically set by Admintools if you set `EnableSSL=1` in the previous version.
- If your `server.crt` SSL certificate file includes certificate chain (more than one certificate), Admintools accepts the whole chained certificate.

7. [Configure SSL](#).

TLS Authentication

Server authentication methods define how clients connect to a Vertica server. Before you define a TLS authentication method, you should understand what type of authentication methods your Vertica server supports. You should also perform any prerequisite tasks.

In regards to SSL, your server can operate with:

- No SSL
- SSL Server Mode —The client does not need certificate or key files.
- SSL Mutual Mode —The client needs certificate, key, and certificate authority files.

SSL modes are independent of authentication, except that the SSL client self-authentication method requires that your server be set-up in SSL Mutual Mode. Otherwise, if you are not implementing client self-authentication method, you can use TLS authentication with either SSL Server Mode or SSL Mutual Mode.

Before you create a TLS authentication method, perform the pre-requisite tasks necessary for your specific environment (for example, certificate creation). Refer to [TLS/SSL Server Authentication](#) and all subsections applicable to your environment.

To create a TLS authentication method, use the command `CREATE AUTHENTICATION` as documented in the SQL Reference Manual.

Implementing Client Self-Authentication

To use a client self-authentication method, your server must be in SSL Mutual Mode.

To create an authentication method for client self-authentication, use the `CREATE AUTHENTICATION` statement. Specify the `auth_type 'tls'` and with `HOST TLS`.

Important: You use the `'tls'` `auth_type` only when you want to create an authentication method for client self-authentication. You must use the `'tls'` `auth_type` with the `HOST TLS` syntax.

Create an Authentication Method with Client Self-Authentication Method

This section provides sample chronological steps for setting up a client for self-authentication, creating an authentication method, and associating the method with a user through a grant statement.

1. Follow all applicable procedures for implementing SSL and distributing certificates and keys. Refer to [TLS/SSL Server Authentication](#) as it applies to your environment.

When you create a client key, make sure to include a Common Name (CN) that is the database user name you want to use with the target database.

```
$ Common Name <server hostname> []:<database username>
```

2. Create the authentication method. Authentication methods are automatically enabled when you create them.

```
=> CREATE AUTHENTICATION myssltest METHOD 'tls' HOST TLS '10.0.0.0/23;
```

3. Associate the method with the user through a grant statement.

```
=> GRANT AUTHENTICATION myssltest TO mydatabaseusername;
```

Your client can now log on and be recognized.

For information on creating authentication methods, refer to the SQL Reference Manual topic, [CREATE AUTHENTICATION](#).

Specify TLS for Client Connections

You can require clients to use TLS when connecting to Vertica. To do so, create a client authentication method for them that uses the HOST TLS syntax with the CREATE AUTHENTICATION statement.

Specific clients might connect through a network connection known to be insecure. In such cases, you can choose to limit specific users to connecting through TLS. You can also require all clients to use TLS.

See [Creating Authentication Records](#) for more information about creating client authentication methods.

LDAP Link Service

LDAP Link enables synchronization between the LDAP and Vertica servers. This eliminates the need for you to manage two sets of users and groups or roles, one on the LDAP server and another on the Vertica server. With LDAP synchronization, the Vertica server becomes a replication database for the LDAP server.

Automatic Synchronization

With LDAP Link the Vertica server closely integrates with an existing directory service such as MS Active Directory or OpenLDAP. The Vertica server automatically synchronizes:

- LDAP users to Vertica users
- LDAP groups to Vertica roles

You manage all user and group properties in the LDAP server. If you are the Vertica database administrator, you need only to set up permissions for Vertica Analytics Platform access on the users and groups.

Configure LDAP Link with LDAP Link connection parameters that reside in the catalog. See [Set LDAP Link Parameters](#) for more information.

Enable LDAP Link

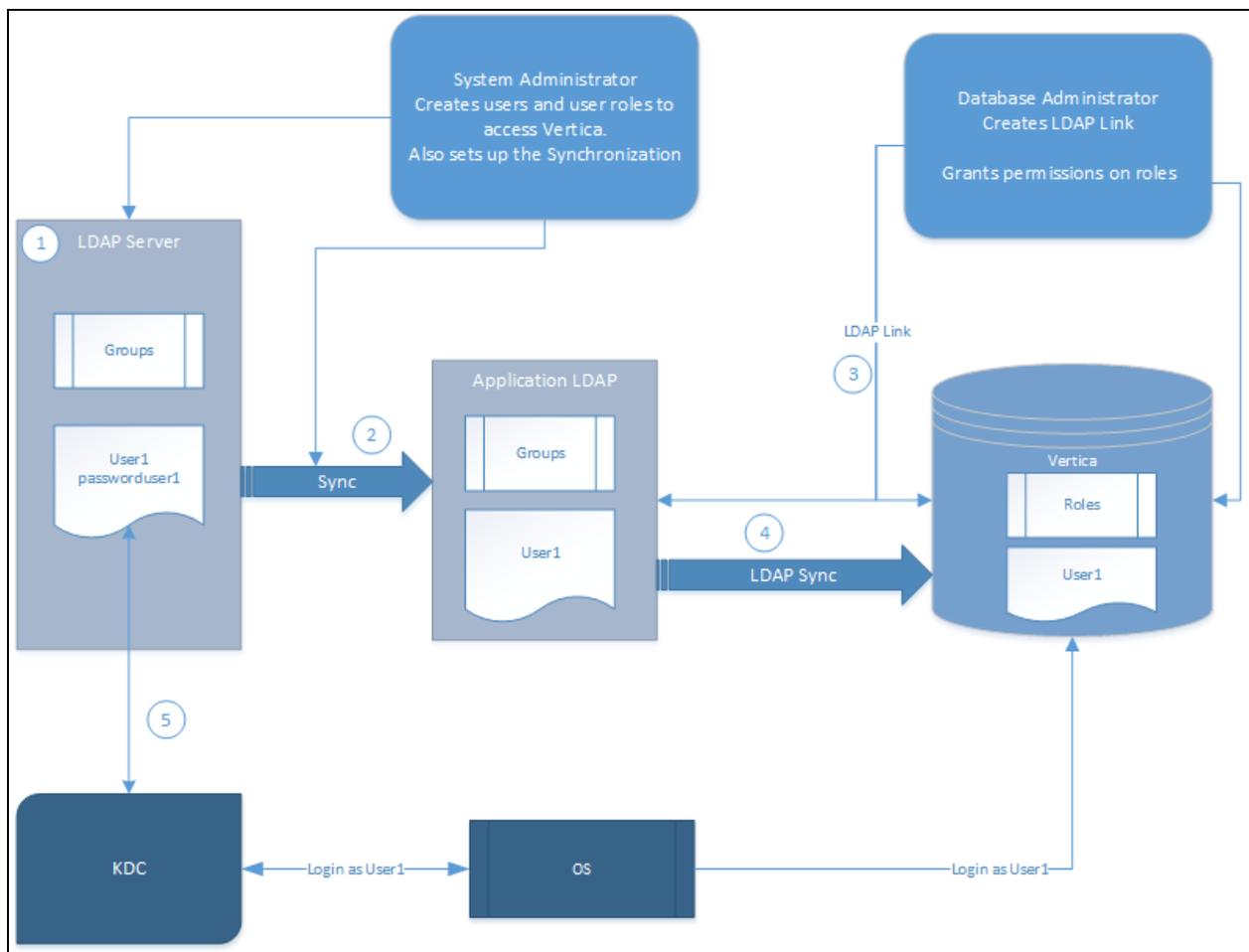
Enable LDAP Link as shown:

```
=> ALTER DATABASE dbname SET PARAMETER LDAPLinkURL='ldap://example.dc.com',  
    LDAPLinkSearchBase='dc=DC,dc=com', LDAPLinkBindDN='CN=jsmith,OU=QA,DC=dc,DC=com',  
    LDAPLinkBindPswd='password',LDAPLinkFilterUser='(objectClass=inetOrgPerson)',  
    LDAPLinkFilterGroup='(objectClass=group)', LDAPLinkOn=1;  
=> SELECT ldap_link_sync_start();
```

See [LDAP Link Parameters](#).

LDAP Link Workflow

After you enable LDAP Link, synchronization occurs according to this workflow:



1. The System Administrator creates users and user groups on the LDAP server.
2. The System Administrator sets up LDAP Link service parameters as required and enables the service.
3. Using the LDAP Link service, Vertica Analytics Platform replicates the users and user groups from the Application LDAP to the Vertica database, creating Vertica users and roles.
4. The LDAP server uses Kerberos (KDC) to authenticate the user logging in to Vertica.
 - The LDAP user can log into Vertica if assigned the appropriate authentication type.
 - After login, you can grant users privileges using GRANT statements or as part of a Group.

Note: After synchronization the Vertica Analytics Platform user does not have an associated authentication method. To allow the user to login, you must assign an authentication method to the user. See [Implementing Client Authentication](#).

Using LDAP Link

When you implement LDAP Link the following are directly affected and help you manage and monitor the LDAP Link - Vertica Analytics Platform synchronization:

- User and Group management
- LDAP Link User Flag
- Blocked Commands
- Client Authentication types

User and Group Management

Users and groups created on the LDAP server have a specific relationship with those users and roles replicated to the Vertica server:

- The user-group relationship on the LDAP server is maintained when those users and groups (roles) are synchronized with Vertica Analytics Platform.
- If a user or group name exists on the Vertica database and a user or group with the same names is synchronized from the LDAP Server using LDAP Link, the users or groups become conflicted. Vertica can not support multiple users with the same name. For a resolution of this conflict, see [User Conflicts](#).

LDAP Link uses the entries in the dn: section of the LDAP configuration file as the unique user identifier when synchronizing a user to the Vertica Analytics Platform:

```
dn: cn=user1,ou=dev,dc=example,dc=com
cn: user1
ou: dev
id: user1
```

The uid parameter in the LDAP configuration file indicates the LDAP user name.

```
uid: user1
```

Upon synchronization, the dn: entry gets mapped to the uid: to identify the Vertica Analytics Platform user.

If you change a setting in the `dn:` and do not change the `uid:` LDAP Link interprets the user as a new user when re-synchronizing with the Vertica Analytics Platform. In this case, the existing Vertica Analytics Platform user with that `uid:` gets deleted from Vertica and a new Vertica Analytics Platform user is created.

If you change the `uid:` and not the `dn:` on LDAP, the `uid` on the Vertica Analytics Platform gets updated to the new `uid`. Since you did not change the `dn:` LDAP Link does not interpret the user as a new user.

LDAP Link User Flag

As a `dbadmin` user, you can access the `vs_users` table to monitor user behavior on the Vertica Analytics Platform. The `users` table contains an `ldap_dn` field that identifies whether or not the Vertica Analytics Platform user is also an LDAP Link user. This example shows the `ldap_dn` field set to `dn` indicating the Vertica Analytics Platform user is also an LDAP Link user:

```
=> SELECT * FROM vs_users;
-[ RECORD 1 ]-----+-----
user_id          | 45035996273704962
user_name        | dbadmin
is_super_user    | t
profile_name     | default
is_locked        | f
lock_time        |
resource_pool    | general
memory_cap_kb    | unlimited
temp_space_cap_kb | unlimited
run_time_cap     | unlimited
max_connections  | unlimited
connection_limit_mode | database
idle_session_timeout | unlimited
all_roles        | dbduser*, dbadmin*, pseudosuperuser*
default_roles    | dbduser*, dbadmin*, pseudosuperuser*
search_path      |
ldap_dn          | dn
ldap_uri_hash    | 0
is_orphaned_from_ldap | f
```

Blocked Commands

Be aware that the following SQL statements are blocked for Vertica Analytics Platform users with `ldapdn` set to `dn` in the `vs_users` table:

- [DROP USER](#) and [DROP ROLE](#)
- [ALTER ROLE RENAME](#)

- [ALTER USER](#) name IDENTIFIED BY 'password' [REPLACE 'old_password']
- [ALTER USER](#) name PASSWORD EXPIRE
- [ALTER USER](#) name PROFILE
- [ALTER USER](#) name SECURITY_ALGORITHM...
- [ALTER USER](#) name DEFAULT ROLE role-name
- [GRANT \(Role\)](#)

Client Authentication Types

LDAP user and groups cannot log into Vertica if client authentication is not assigned to the user or group. You can use the following valid [authentication types](#) for LDAP users and groups:

- GSS
- Ident
- LDAP
- Reject
- Trust

LDAP Link Parameters

Use LDAP Link parameters to determine:

- LDAP Link operations, such as enabling or disabling LDAP Link and how often to perform replication
- Authentication parameters, including SSL authentication parameters
- Users and groups that inherit unowned objects
- How to resolve conflicts

Set LDAP Link Parameters

This example shows how you can set:

- The URL of the LDAP server (LDAPLinkURL) and
- The base DN from where to start replication (LDAPLinkSearchBase)

You also see how to set the LDAP Link Bind authentication parameters (LDAPLinkBindDN and LDAPLinkBindPswd) and enables LDAP Link (LDAPLinkOn).

```
=> ALTER DATABASE myDB1 SET PARAMETER LDAPLinkURL='ldap://10.60.55.128',  
LDAPLinkSearchBase='dc=corp,dc=com',LDAPLinkBindDN='dc=corp,dc=com',LDAPLinkBindPswd='password';  
  
=> ALTER DATABASE myDB1 SET PARAMETER LDAPLinkOn = '1';
```

General and Connection Parameters

Parameter	Description
LDAPLinkOn	Enables or disables LDAP Link. Valid Values: 0 —LDAP Link disabled 1 —LDAP Link enabled Default value: 0
LDAPLinkURL	The LDAP server URL. Example: <pre>SET PARAMETER LDAPLinkURL='ldap://glw2k8-64.dc.com';</pre>
LDAPLinkInterval	The time interval, in seconds, by which the LDAP Server and Vertica server synchronize. Default Value: 86400 (one day).
LDAPLinkFirstInterval	The first interval, in seconds, for LDAP/Vertica synchronization after the clerk node joins the cluster. Default Value: 120

Parameter	Description
LDAPLinkRetryInterval	The time, in seconds, the system waits to retry a failed synchronization. Default Value: 10
LDAPLinkRetryNumber	The number of retry attempts if synchronization failed. Default Value: 10.
LDAPLinkSearchBase	The base dn from where to start replication. Example: <pre>SET PARAMETER LDAPLinkSearchBase='ou=vertica,dc=mycompany,dc=com';</pre> Vertica recommends using a separate OU for database users.
LDAPLinkSearchTimeout	The timeout length, in seconds, for the LDAP search operation during an LDAP Link Service run. Default Value: 10
LDAPLinkScope	Indicates what dn level to replicate. Valid Values: <ul style="list-style-type: none"> • sub—Replicate entire subtree under baseDN • one—Replicate to one level under baseDN • base —Replicate only the baseDN level <p>If you decrease the scope (for example, sub to one), some users may not be recognized during the next synchronization.</p> Default Value: sub
LDAPLinkFilterUser	Determines how to filter users to be replicated. Default Value: "(objectClass=inetOrgPerson)"
LDAPLinkFilterGroup	Determines how to filter groups to be replicated. Default Value: "(objectClass=groupofnames)"
LDAPLinkGroupName	[Optional] The LDAP field to use when creating a role name in Vertica.

Parameter	Description
	Default Value: cn
LDAPLinkGroupMembers	The LDAP group name linked to a user. Default Value: member
LDAPLinkUserName	The LDAP field to use when creating a user name in Vertica.

Authentication Parameters

Parameter	Description
LDAPLinkBindDN	The LDAP Bind DN used for authentication. Example: <pre>SET PARAMETER LDAPLinkBindDN='CN=amir,OU=QA,DC=dc,DC=com';</pre>
LDAPLinkBindPswd	The valid password for the LDAP Bind DN to access the server. Only accessible by the dbadmin user. Example: <pre>SET PARAMETER LDAPLinkBindPswd='password';</pre>

SSL Authentication Parameters

Parameter	Description
LDAPLinkStartTLS	[Optional] Specifies whether or not to use the StartTLS operation during bind. You can only use this parameter if the LDAP server's URL is "ldap://..." (not "ldaps://...") Valid Values: 0 - Do not use starttls 1 - Use starttls Default Value: 0
LDAPLinkTLSReqCert	[Optional] Specifies how Vertica behaves when verifying

Parameter	Description
	<p>the LDAP server's certificate when using TLS. The connection between Vertica and the LDAP server will only succeed if conditions for the specified value are met.</p> <p>Valid Values:</p> <p>hard: LDAP server must provide a valid certificate.</p> <p>allow: LDAP server does not have to provide a certificate.</p> <p>try: LDAP server must either provide a valid certificate or not provide one at all.</p> <p>never: No requirements (Vertica does not request the LDAP server's certificate).</p> <p>For details, see Using LDAP Over SSL/TLS.</p> <p>Default Value: allow</p>
LDAPLinkTLSCACert	[Optional] The path to the CA certificates.

Miscellaneous Parameters

Parameter	Description
LDAPLinkConflictPolicy	<p>Determines how to resolve a user conflict.</p> <p>Valid Values:</p> <p>IGNORE—Ignores the incoming LDAP user and maintains the existing Vertica user.</p> <p>MERGE—Converts the existing user to an LDAP user.</p> <p>Default Value: MERGE</p>
LDAPLinkStopIfZeroUsers	<p>Enables or disables the shutdown of LDAPLink synchronization if no users are found in LDAP.</p> <p>Valid values:</p> <p>0 - Disables the LDAPLink synchronization shutdown if no users are found. This may lead to inadvertent dropping of Vertica users.</p> <p>1 - Enables the LDAPLink synchronization shutdown if no</p>

Parameter	Description
	users are found. This prevents inadvertent dropping of Vertica users.
LDAPLinkDryRun	<p>[Optional] Tests the connection to the LDAP server and logs the response without doing a synchronization. Also tests if parameters are correctly set.</p> <p>Valid Values:</p> <p>0 - Disables LDAPLinkDryRun</p> <p>1 - Enables LDAPLinkDryRun</p> <p>Default Value: 0</p>

See [Managing Configuration Parameters: VSQL](#) for information on setting LDAP Link parameters.

Note: When you change any Connection or Authentication parameter, LDAP Link reconnects and re-initializes the synchronization.

Troubleshooting LDAP Link Issues

Various issues can arise with LDAP Link Service, including:

- Disconnected (Orphaned) Users and Roles
- Lost Objects
- User Conflicts

Disconnected (Orphaned) Users and Roles

Vertica Analytics Platform users and roles synchronized through LDAP Link can become disconnected, or *orphaned*, if an issue arises with the LDAP Link service. For example, users and roles become orphaned when you change the connection to the LDAP server as the following scenario describes:

1. Create an LDAP connection as follows:

```
=> ALTER DATABASE MyDB1 SET PARAMETER LDAPLinkURL='ldap://ebuser',  
LDAPLinkSearchBase='dc=example,dc=com', LDAPLinkBindDN='mega',  
LDAPLinkBindPswd='$megapassword$';  
=> ALTER DATABASE MyDB1 SET PARAMETER LDAPLinkOn = '1';
```

2. Run an LDAP Link session to synchronize LDAP and Vertica users.
3. Change one or more connection parameters from Step 1. You can change the connection only if you change one of the LDAPLinkURL or LDAPLinkSearchBase parameters.
4. Run another LDAP Link session. The system attempts to re-synchronize LDAP and Vertica users. Since the connection has changed, the existing Vertica users cannot be synchronized with the LDAP users from the new connection. These Vertica users become orphaned.

As a dbadmin user, you can identify orphaned users by checking the field `is_orphaned_from_ldap` in the `users` system table:

```
=> SELECT is_orphaned_from_ldap FROM users;
```

A field value of `t` indicates that the user is an orphaned user. Orphaned Vertica users cannot connect to the LDAP server and cannot login to Vertica using LDAP authentication (however, other authentication methods assigned to the user work). In this case, you can delete the orphaned Vertica user and run the LDAP Link service to resynchronize users.

Re-parented Objects

When you delete users or groups from linked LDAP, the LDAP Link service removes the same users and roles from Vertica Analytics Platform. However, the service does not delete objects owned by the deleted user. Use the `GlobalHeirUserName` parameter to assign the objects to a new owner (re-parent).

Example:

```
=> ALTER DATABASE example_db SET PARAMETER GlobalHeirUserName=user1;
```

This creates a new user named `user1`, if it does not exist. The `GlobalHeirUsername` user serves as the new parent for all the objects owned by deleted users.

By default, this parameter is set to `<auto>` which re-parents the objects to the `dbadmin` user.

If you leave `GlobalHeirUsername` empty, the objects are not re-parented to another user.

For more information see [GlobalHeirUserName](#) in Security Parameters.

User Conflicts

Vertica Analytics Platform users and roles synchronized using LDAP Link can become conflicted. Such conflicts can occur, for example, when you create a new user or group on the LDAP server and another user or role with the same name exists on the Vertica Analytics Platform.

As a dbadmin user, use one of the following parameters to resolve user conflicts:

- `LDAPLinkConflictPolicy`
- `LDAPLinkStopIfZeroUsers`

LDAPLinkConflictPolicy

Use `LDAPLinkConflictPolicy` to resolve any user conflicts:

- `LDAPLinkConflictPolicy=IGNORE` - Ignores the incoming LDAP users and maintains the existing Vertica user
- `LDAPLinkConflictPolicy=MERGE` - Merges the incoming LDAP user with the Vertica user and converts the database user to an LDAP user retaining the database user's objects

Example:

```
=> ALTER DATABASE example_db SET PARAMETER LDAPLinkConflictPolicy='MERGE';
```

The default is `MERGE`. If you change `LDAPLinkConflictPolicy`, the change takes effect on the next synchronization.

LDAPLinkStopIfZeroUsers

Use `LDAPLinkStopIfZeroUsers` to prevent an accidental dropping of Vertica users if the LDAP Link synchronization does not find any LDAP users.

`LDAPLinkStopIfZeroUsers=0` - Does not stop the LDAP Link synchronization if no users are found in LDAP. This drops all Vertica users during synchronization.

`LDAPLinkStopIfZeroUsers=1` - Stops the LDAP Link synchronization if no users are found in LDAP and displays an error. This prevents the dropping of Vertica users due to some issue.

Monitoring LDAP Link

Use the `ldap_link_events` table to monitor events that occurred during an LDAP Link synchronization:

```
=> SELECT transaction_id, event_type, entry_name, entry_oid FROM ldap_link_events;
  transaction_id | event_type | entry_name | entry_oid
-----+-----+-----+-----
45035996273705317 | SYNC_STARTED | | 0
45066962732553589 | SYNC_FINISHED | | 0
45066988112255317 | PROCESSING_STARTED | | 0
23411234566789765 | USER_CREATED | tuser | 234548899
(4 rows)
```

Connector Framework Service

The Connector Framework Service (CFS) allows secure indexing of documents from IDOL to the Vertica Analytics Platform. Access Control Lists determine which users have permissions to access documents. Documents transferred from IDOL are stored in a Flex table ([Using Flex Tables](#)).

The dbadmin creates a view from this flex table (see [Views](#)). Users access the IDOL data from these views.

CFS Components

Use the following CFS components to implement the service on the Vertica:

- IDOL document metadata
- Security key and authorization functions
- SQL statement and Security Parameter
- CFS Configuration file

See [Implementing CFS](#)

Document Metadata

Vertica Analytics Platform stores IDOL document metadata in a flex table. Set the name of the flex table with the TableName parameter in the CFS configuration file (see [Modify the CFS Configuration File](#)). The metadata includes the following:

- AUTONOMYMETADATA (Mandatory)—An alphanumeric designation for the ACL designated for the document.
- DREFIELD—Assigns permission levels to users and groups for accessing IDOL documents.
- DRETITLE—The document title.

For information see .

Security Key and Authorization Functions

As the dbadmin user you can assign each user a unique Security Information String (SIS). The strings are encrypted with a key stored in `/idol/community/key`. This key uses the following functions:

- [INSTALL_COMMUNITY_KEY](#)
- [DESCRIBE_COMMUNITY_KEY](#)
- [DELETE_COMMUNITY_KEY](#)

In addition to the preceding functions, the authorization function [IDOL_CHECK_ACL](#) verifies that the user has access to data in the view.

These functions reside in the `v_idol` schema contained in the `idollib` library installed with the CFS. When you run the functions, you must use `v_idol` in the command, for example:

```
=> SELECT v_idol.DELETE_COMMUNITY_KEY();
```

SQL Statement and Security Parameter

As the dbadmin user set the Security Information String for a specific user using:

- [ALTER SESSION](#)
- `IdolSecurityInfo` user-defined session parameter.

CFS Configuration File

You must index IDOL metadata in Vertica Analytics Platform to be available for queries. See [Implementing CFS](#)

Implementing CFS

After Vertica Analytics Platform ingests documents from IDOL, you can implement CFS to secure those documents. Implementing the security requires action from both the Vertica database administrator and the user who runs queries.

The database administrator must:

- [Modify the CFS Configuration File](#)
- [Create a View](#) from the flex table where the ingested IDOL data resides
- [Install the Security Key](#) used to decrypt the Security Information String
- [Verify CFS implementation success](#)

Upon CFS implementation, the user can [Query the IDOL Data](#):

Modify the CFS Configuration File

Set the following in the CFS configuration file to have CFS automatically index the metadata:

1. In the [Indexing] section, set the IndexerSections parameter to vertica:

```
[Indexing]
IndexerSections=vertica
IndexBatchSize=1
IndexTimeInterval=30
```

2. Create a new section with the same name you entered in the IndexerSections parameter and enter the following parameters and keywords:

```
[vertica]
IndexerType=Library
ConnectionString=Driver=Vertica;Server=123.456.478.900;Database=myDb;UID=dbadmin;PWD=password
TableName=myFlexTable
LibraryDirectory= ./shared_library_indexers
LibraryName=VerticaIndexer
```

The verticalIndexer (LibraryName above) is part of CFS. To use this tool, you must install and configure the Vertica ODBC drivers on the same machine as CFS. CFS sends JSON-formatted data to the Flex table using ODBC. For more information, see [Installing ODBC Drivers on Linux, Solaris, AIX, and HP-UX](#).

Create a View

The Vertica database administrator must create a view from the flex table where the ingested IDOL data resides. Users querying IDOL data do so using this view, thus preventing unauthorized access to the flex table.

This example creates a view from the `idol_table`:

```
=> CREATE VIEW idol_view as select * from idol_table where v_idol.idol_check_acl(acl, security_
section, security_type);
CREATE VIEW
```

You must then run GRANT privileges as follows:

1. Grant usage on the `v_idol` schema:

```
=> GRANT USAGE on schema v_idol to user1;
GRANT PRIVILEGE
```

2. Grant SELECT privileges on the view to any user who needs access:

```
=> GRANT SELECT on idol_view to user1;
GRANT PRIVILEGE
```

3. Grant usage on the library. This allows you to set UDSession parameters later:

```
=> GRANT USAGE on library v_idol.IdolLib to user1;
GRANT PRIVILEGE
```

4. Grant execute privileges on the acl-checking function.

```
=> GRANT EXECUTE on function v_idol.idol_check_acl(long varchar, long varchar, long varchar) to
user1;
GRANT PRIVILEGE
```

Important: When granting EXECUTE privileges on a CFS function, ensure that you only grant the privilege to `v_idol.idol_check_acl()`. Never grant EXECUTE privileges to the following CFS functions:

```
v_idol.install_community_key()
v_idol.describe_community_key()
v_idol.delete_community_key()
```

Install the Security Key

Before a user can access the view containing IDOL data, the database administrator must retrieve and install the security key. Store the security key in Vertica's Distributed File System (DFS) with the following command:

```
=> SELECT v_idol.install_community_key(USING PARAMEERS file_
path='/home/user1/Downloads/IDOL/cli/aes.txt');
```

The Security Key is the same one used in IDOL. You can only install one key at a time. Installing another key overwrites the existing one.

The SecurityInfoKeys parameter in the following IDOL configuration files indicates the key you need to install:

- community/Community.cfg
- content/Content.cfg

After installing the key, run the following command to ensure it was correctly installed :

```
=> SELECT v_idol.describe_community_key();
           describe_community_key
-----
daf3cc7d3c9368c2e78ed336aec6d8e75ee038c47ajc76f86a3bb5b197639ed43725a
(1 row)
```

See [DELETE_COMMUNITY_KEY](#) for information on deleting security keys.

Verify CFS implementation success

As a database administrator, do the following to verify the successful implementation of CFS:

1. Retrieve the Security Information String (SIS) from your organization's application. This is accomplished outside of Vertica.
2. Set the retrieved SIS for your current session:

```
=> ALTER SESSION set UDPARAMETER FOR v_idol.IdolLib IdolSecurityInfo =
'MzA0U08/+T59PHj6Pn64v/m8A';
```

3. Run the following ACL check function to see if it returns true only for rows to which you should have access:

```
=> SELECT v_idol.idol_check_acl(acl, security_section, security_type) from cfs_table;
      idol_check_acl
-----
                f
                t
(2 rows)
```

Alternatively you can run the ACL check function and explicitly pass the SIS:

```
=> SELECT v_idol.idol_check_acl(acl, security_section, security_type using parameters
sis='MzA0U08/+T59PHj6Pn64v/m8A')
FROM cfs_table;
      idol_check_acl
-----
                f
                t
(2 rows)
```

Query the IDOL Data

A user querying IDOL must do the following:

1. Retrieve the Security Information String (SIS) from your organization's application. This is accomplished outside of Vertica.
2. Set the retrieved SIS for your current user session:

```
=> ALTER SESSION set UDPARAMETER FOR v_idol.IdolLib IdolSecurityInfo =
'MzA0U08/+T59PHj6Pn64v/m8A';
```

This example checks the `v_idol.t` flex table for the idol access control list, security section, and security type. All these parameters must match the metadata in the flex table.

3. Run the query:

```
=> SELECT * FROM idol_view;
```

Federal Information Processing Standard

When running on a certified FIPS-140-2 Red Hat 6.6 system, Vertica uses a certified OpenSSL FIPS 140-2 cryptographic module. This meets the security standards set by the National Institute of Standards and Technology (NIST) for Federal Agencies in the United States or other countries.

The standard specifies the security requirements that a cryptographic module needs in a system protecting sensitive information. For details on the standard see the [Computer Security Resource Center](#).

Note: Vertica itself is not FIPS compliant but it is compatible with running on a FIPS-enabled system using FIPS resources.

For a list of FIPS prerequisites, see [FIPS 140-2 Supported Platforms](#).

OpenSSL Behavior

Dynamic OpenSSL linking is a requirement for a FIPS implementation on the client and server. The Vertica server uses the OpenSSL that resides on the host system (version 1.0.1e as indicated in [FIPS 140-2 Supported Platforms](#)). OpenSSL dynamically links with LDAP and Kerberos.

For more information see [Locate OpenSSL Libraries](#).

Libraries on CentOS 6.6 FIPS Systems

On a CentOS 6.6 FIPS system, Vertica runs only with the OpenSSL libraries `libcrypto.so.1.0.1e` and `libssl.so.1.0.1e`. Other versions of these libraries do not run on a FIPS system. This incompatibility occurs because the FIPS security policy checksums the library to which an application is linked and verifies that the library the application executes with the same checksum.

Library Versioning on Non-FIPS Systems

Be aware that on some non-FIPS systems, versioning anomalies can occur when you install a new version of OpenSSL. Sometimes, the default OpenSSL build procedure produces libraries with versions named 1.0.0. For Vertica to recognize that a library has a higher version number, you must provide the library name with a higher version number. For example, when installing OpenSSL version 1.0.1t, name the libraries `libcrypto.so.1.0.1t` or `libssl.1.0.1t` (symbolic links with these names are sufficient).

Install FIPS-enabled Vertica

The Vertica Analytics Platform installation process determines if your system environment is FIPS compliant by checking the file `/proc/sys/crypto/fips_enabled` as follows:

```
$ sysctl crypto.fips_enabled  
crypto.fips_enabled = 1
```

- If `fips_enabled` contains a 1, the host is FIPS enabled.
- If `fips_enabled` contains a 0, the host is not FIPS enabled.

If the host is FIPS enabled, the installation does the following:

- Verifies that OpenSSL resides in the appropriate area. If application does not exist before installation, the installer uses the OpenSSL provided by Vertica as the default. (OpenSSL is stored in `/opt/vertica/lib`).
- Runs a test to verify that Vertica was successfully configured for FIPS. If this test fails on any node, the installer fails.

For more information see [Installing Vertica](#).

FIPS-Enabled Databases

Manually creating a new database on a FIPS-enabled Vertica Analytics Platform requires a different approach than for a non-FIPS machine. Be aware of the following limitations:

- You cannot create a FIPS-enabled database on a non-FIPS machine.
- You cannot create a non-FIPS database on a FIPS-enabled machine.
- Copying data generated with the MD5 hashing algorithm from a non-FIPS machine to a FIPS-enabled machine results in data corruption.

Implementing FIPS 140-2

Implementing FIPS 140-2 on your Vertica Analytics Platform requires configuration on the server and client. The Vertica server uses FIPS-approved algorithms; however Vertica clients may be running on non-FIPS-approved systems. Therefore, you must implement FIPS 140-2 compliance from end to end.

For more information on implementing FIPS, see:

- [Implement FIPS on the Server](#)
- [Implement FIPS on the Client](#)

Implement FIPS on the Server

To implement FIPS on the Vertica server, you must:

- Generate a secure SSL certificate to establish a secure connection to the client.
- If necessary, set the `LD_LIBRARY_PATH` environment variable to locate the OpenSSL libraries.

RequireFIPS Parameter

Upon startup Vertica sets the `RequireFIPS` parameter on the server to reflect the FIPS state of the system, as follows:

Specify that FIPS is disabled:

```
RequireFIPS = 0
```

Specify that FIPS is enabled:

```
RequireFIPS = 1
```

The value of RequireFIPS matches the value of `crypto.fips_enabled` file. See [Install FIPS-enabled Vertica](#).

Verify the value of the RequireFIPS parameter as follows:

```
=> SELECT get_config_parameter('RequireFIPS');
       get_config_parameter
-----
0
(1 row)
```

Depending on the FIPS state, the following behaviors can occur:

- If the file `/proc/sys/crypto/fips_enabled` exists and contains a 1 (FIPS-enabled), Vertica sets RequireFIPS to 1. Modifying RequireFIPS with `ALTER DATABASE (RequireFIPS = 0)` generates an error.
- If the file `/proc/sys/crypto/fips_enabled` does not exist, or exists and contains a 0 (non-FIPS), Vertica automatically sets RequireFIPS to 0. Modifying RequireFIPS with `ALTER DATABASE (RequireFIPS = 1)` generates an error.
- If the FIPS state of a node, as determined from the existence of `/proc/sys/crypto/fips_enabled`, differs from the state received from the cluster initiator, the node fails. This behavior prevents the creation of clusters of mixed FIPS and non-FIPS systems.

Important: If you attempt to restore a FIPS-enabled node to a non-FIPS cluster, the restore process fails.

Locate OpenSSL Libraries

Vertica must find and load the correct OpenSSL libraries, `libcrypto.so.1.0.1.e` and `libssl.so.1.0.1.e`. To do so, it searches the system directory where the libraries reside. If the SSL libraries are not found, Vertica uses its own openssl libraries that reside under `/opt/vertica/lib`.

Note: If you do not use admintools to start Vertica, or have conflicting libraries in your system, you must manually set LD_LIBRARY_PATH with /opt/vertica/lib appearing first in the list. When admintools starts or reboots Vertica, the path is set automatically.

Secure Client-Server Connection

Vertica uses TLS 1.2 to support the server-client connection for a FIPS-enabled system. This specification includes using a server certificate issued by a Certificate Authority.

Note: Using TLS 1.2 prevents you from using the MD5 algorithm for hashing passwords. Vertica accepts only AuthenticatedClearTextPasswords hashed by SHA-512. Users with MD5 passwords must migrate to SHA-512 passwords. For more information, see [Upgrade Considerations for Hash Authentication](#).

For instructions on generating a self-signed certificate see [Generating SSL Certificates and Keys](#).

After generating a certificate, you need to distribute it to all hosts on the cluster. See [Distributing Certificates and Keys](#). This distribution stores the certificate in the SSLCertificate parameter and the private key in the SSLPrivateKey parameter. For more information see [Security Parameters](#).

Implement FIPS on the Client

Vertica provides a FIPS-compliant client driver, which you can install on a FIPS-enabled system. The 64-bit client includes vsql and ODBC drivers.

For information on the FIPS client, and installation, refer to the following

- [Installing the FIPS Client Driver for ODBC and vsql](#).
- [Installing the FIPS Client Driver for JDBC](#)

FIPS 140-2 Compliance Statement

Contents

1. [Summary](#)
2. [Overview](#)

[a. About Vertica](#)

[b. About FIPS 140-2](#)

[3. Vertica and FIPS 140-2](#)

1. Summary

Vertica complies with Federal Information Processing Standard 140-2 (FIPS 140-2), which defines the technical requirements to be used by Federal Agencies when these organizations specify cryptographic-based security systems for protection of sensitive or valuable data. The compliance of Vertica with FIPS 140-2 is ensured by: 1) Integrating validated and NIST-certified third party cryptographic module(s), and using the module(s) as the only provider(s) of cryptographic services; 2) Using FIPS-approved cryptographic functions; 3) Using FIPS-approved and NIST-validated technologies applicable for Vertica design, implementation and operation.

2. Overview

a. About Vertica

- Vertica is a high performance relational database management system used for advanced analytics applications. Its performance and scale is achieved through a columnar storage and execution architecture that offers a massively parallel processing solution. Aggressive encoding and compression allows Vertica analytics to perform by reducing CPU, memory and disk I/O Processing times.
- For more details about Vertica and its usage, see [Vertica Concepts](#).

b. About FIPS 140-2

FIPS (Federal Information Processing Standard) 140-2, *Security requirements for cryptographic modules*, is the Federal standard for proper cryptography for computer systems purchased by the government.

The Federal Information Processing Standards Publication (FIPS) 140-2, “Security Requirements for Cryptographic Modules,” was issued by the National Institute of Standards and Technology (NIST) in May, 2001.

The benefits of using FIPS 140-2 validated crypto module is that the crypto algorithms are deemed appropriate and that they perform the encrypt/decrypt/hash functions correctly. The standard specifies the security requirements for cryptographic modules utilized within a

security system that protects sensitive or valuable data. The requirements can be found in the following documents:

- [SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES](#)
- [Annex A: Approved Security Functions for FIPS PUB 140-2, Security Requirements for Cryptographic Modules](#)

3. Vertica and FIPS 140-2

FIPS 140-2 validated third party module

Vertica conforms with FIPS 140-2 Level 1 compliance by dynamically linking to the FIPS 140-2 approved OpenSSL cryptographic module provided by the Operating System, which in our initial release is [RedHat Enterprise Linux 6.6 OpenSSL Module](#).

Vertica can be configured to operate in FIPS-compliant mode ensuring its functions and procedures like SSL/TLS connections, which require cryptography (secure hash, encryption, digital signatures, etc.) makes use of the crypto services provided by RedHat Enterprise 6.6 OpenSSL module Version 3.0 which is validated for FIPS 140-2. If you are not running on RedHat 6.6 you will not be able to run Vertica on FIPS mode. The assurance that Vertica is using the right FIPS 140-2 encryption modules is managed at the operating system level by RedHat's implementation.

Vertica checks the OS level flag setting `/proc/sys/crypto/fips_enabled` to kick off Vertica's FIPS mode installation. Further details about how to install and configure Vertica and its components to conform to FIPS 140-2 standard appear in the installation and security guides:

- [Installing Vertica with the Installation Script](#)
- [Federal Information Processing Standard](#)

Modes of Operation

Vertica Server operates in one of two modes determined by the OS configuration.

- FIPS-compliant mode – supports FIPS 140-2 compliant cryptographic functions. In this mode, all cryptographic functions, default algorithms and key lengths are bound to those allowed by FIPS 140-2.
- Standard mode – non-FIPS 140-2 compliant mode which utilizes all existing Vertica cryptography functions.

TLS/SSL3.x

All the Vertica client/server communications can be secured with FIPS-compliant Transport Layer Security TLS1.2/SSL3.1 or higher. It is relying on FIPS 140-2 approved hash algorithms and ciphers.

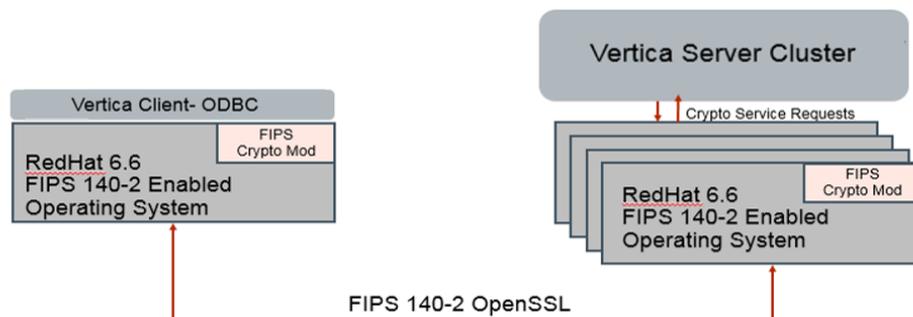
- TLS handshake, key negotiation and authentication provides data integrity and uses secure hash and FIPS 140-2 approved cryptography and digital signature.
- TLS encryption of data in transit provides confidentiality and making use of FIPS 140-2 approved cryptography.

Secure Hash

Per FIPS 140-2 standards, Vertica, in the FIPS 140-2 compliant mode, can be configured to use only the SHA-512 algorithm.

FIPS 140-2 Architecture

Vertica is a relational database system that is comprised of a client component and a server component. On the Client Side, we offer a suite of drivers for host clients to access the Vertica Server Side component. Both client and server Vertica components conform to FIPS 140-2 Level 1 compliance by dynamically linking to the FIPS 140-2 approved OpenSSL cryptographic module provided by RedHat Enterprise Linux 6.6 OpenSSL Module.



Supported platforms

Supported operating systems (both client and server):

- RedHat Enterprise Linux 6.6 64 bit

Supported client drivers:

- ODBC only

Design Assurance

Vertica uses the security provider Red Hat Enterprise Linux 6.6 OpenSSL Module v3.0. This is the only supported security provider for FIPS 140-2.

Once you have configured Vertica to be compliant with FIPS 140-2, you cannot revert back to the standard configuration unless you disable FIPS 140-2 at the operating system level. Please reference the following documentation section for considerations:

- [Implement FIPS on the Server](#)
- [Implement FIPS on the Client](#)

System Table Restriction

The security parameter, `RestrictSystemTables`, prohibits users without database administrator privileges from viewing potentially sensitive information in system tables. For example, as a database administrator you may want to restrict access to the `CONFIGURATION_PARAMETERS` table to prevent any inadvertent modifications.

`RestrictSystemTables` is disabled by default. Enable the `RestrictSystemTables` parameter as follows:

```
=> ALTER DATABASE <database name> SET PARAMETER RestrictSystemTables=1;  
WARNING 4324: Parameter RestrictSystemTables will not take effect until database restart
```

Disable the `RestrictSystemTables` parameter as follows:

```
=> ALTER DATABASE <database name> SET PARAMETER RestrictSystemTables=0;  
WARNING 4324: Parameter RestrictSystemTables will not take effect until database restart
```

Note: You must restart the server to implement any changes to `RestrictSystemTables`.

When you enable `RestrictSystemTables`, the following conditions take affect:

- System tables with built-in restrictions that determine who can access information remain accessible. For example, the `SESSIONS` table allows a user to see only information on that user's current session when `RestrictSystemTables` is enabled.
- System tables that contain settings that a user should be allowed to see are accessible when `RestrictSystemTables` is enabled. For example, the `TYPES` system table.
- System tables that contain information about other users are restricted.
- System tables that contain information about database settings are restricted.
- The behavior of the `SYSMONITOR Role` does not change.

Privileges

You must be a database administrator user to modify `RestrictSystemTables` settings.

Determine if a Table is Restricted

Use the `SYSTEM_TABLES` to determine if a table is accessible. The table contains a column called `is_accessible_during_lockdown`. This flag indicates if a table is accessible when `RestrictedSystemTables` is enabled. Valid values are:

- t — A table is accessible
- f — A table is not accessible

Show Accessible System Tables

The following examples show how you can query system tables that have restrictions.

Query the contents of all the system tables that have the `is_accessible_during_lockdown` flag set for each table:

```
=> SELECT * FROM system_tables;
table_schema_id | catalog
table_name      | sales_projections
table_id        | 834868
table_description | sales forecast 2016
is_superuser_only | f
is_monitorable  | f
is_accessible_during_lockdown | t
```

In the preceding example, the `sales_projection` table remains accessible to all users when `RestrictSystemTables` is enabled.

Query a list of all tables that are accessible when `RestrictSystemTables` is enabled:

```
=> SELECT table_name from system_tables where is_accessible_during_lockdown='t';
-[ RECORD 1 ]-----
table_name | user_audits
-[ RECORD 2 ]-----
table_name | views
-[ RECORD 3 ]-----
table_name | dual
-[ RECORD 4 ]-----
table_name | user_transforms
```

Query a list of all tables that are not accessible when `RestrictSystemTables` is enabled:

```
=> SELECT table_name from system_tables where is_accessible_during_lockdown='f';
-[ RECORD 1 ]-----
table_name | user_client_auth
```

```
-[ RECORD 2 ]-----  
table_name | directed_queries  
-[ RECORD 3 ]-----  
table_name | storage_locations  
-[ RECORD 4 ]-----  
table_name | system_columns
```

Extending Vertica

You can extend Vertica to perform new operations or handle new types of data. There are several types of extensions:

- **External procedures** let you execute external scripts or programs that are installed on a host in your database cluster.
- **User-Defined SQL Functions** let you store frequently-used SQL expressions. They can help you simplify and standardize your SQL scripts.
- **User-Defined Extensions (UDxs)** let you develop your own analytic or data-loading tools using the C++, Python, Java, and R programming languages. They are useful when the type of data processing you want to perform is difficult or slow using SQL.

The following sections explain how to use extensions that have already been written:

- [Using External Procedures](#)
- [Using User-Defined SQL Functions](#)
- [Using User-Defined Extensions](#)

[Developing User-Defined Extensions \(UDxs\)](#) explains how to write new UDxs, including the following types:

- [Aggregate Functions \(UDAFs\)](#)
- [Analytic Functions \(UDAnFs\)](#)
- [Scalar Functions \(UDSFs\)](#)
- [Transform Functions \(UDTFs\)](#)
- [Load \(UDLs\)](#)

Using External Procedures

The external procedure feature lets you call a script or executable program on a host in your database cluster from within Vertica. You can pass literal values to this external procedure as arguments. The external procedure cannot communicate back to Vertica.

This chapter explains how to create, install, and use external procedures.

Implementing External Procedures

To implement an external procedure:

1. Create an external procedure executable file.

See [Requirements for External Procedures](#).
2. Enable the set-user-ID(SUID), user execute, and group execute attributes for the file. The file must either be readable by the dbadmin or the file owner's password must be given with the Administration Tools `install_procedure` command.
3. [Install the external procedure executable file](#).
4. [Create the external procedure in Vertica](#).

Once a procedure is created in Vertica, you can [execute](#) or [drop](#) it, but you cannot alter it.

Requirements for External Procedures

External procedures have requirements regarding their attributes, where you store them, and how you handle their output. You should also be cognizant of their resource usage.

Procedure File Attributes

The procedure file cannot be owned by root. It must have the set-user-ID (SUID), user execute, and group execute attributes set. If it is not readable by the Linux database administrator user, then the owner's password will have to be specified when installing the procedure.

Handling Procedure Output

Vertica does not provide a facility for handling procedure output. Therefore, you must make your own arrangements for handling procedure output, which should include writing error, logging, and program information directly to files that you manage.

Handling Resource Usage

The Vertica resource manager is unaware of resources used by external procedures. Additionally, Vertica is intended to be the only major process running on your system. If your external procedure is resource intensive, it could affect the performance and stability of Vertica. Consider the types of external procedures you create and when you run them. For example, you might run a resource-intensive procedure during off hours.

Sample Procedure File

```
#!/bin/bash  
echo "hello planet argument: $1" >> /tmp/myprocedure.log
```

Installing External Procedure Executable Files

To install an external procedure, use the Administration Tools through either menu or the command line.

Using the Admin Tools Menus

- a. Run the Administration Tools.

```
$ /opt/vertica/bin/adminTools
```

- b. On the AdminTools **Main Menu**, click **Configuration Menu**, and then click **OK**.
- c. On the **Configuration Menu**, click **Install External Procedure** and then click **OK**.
- d. Select the database on which you want to install the external procedure.
- e. Either select the file to install or manually type the complete file path, and then click **OK**.
- f. If you are not the superuser, you are prompted to enter your password and click **OK**.

The Administration Tools automatically create the <database_catalog_path>/procedures directory on each node in the database and installs the external

procedure in these directories for you.

- g. Click **OK** in the dialog that indicates that the installation was successful.

Command Line

If you use the command line, be sure to specify the full path to the procedure file and the password of the Linux user who owns the procedure file;

For example:

```
$ admintools -t install_procedure -d vmartdb -f /scratch/helloworld.sh -p ownerpassword
Installing external procedure...
External procedure installed
```

Once you have installed an external procedure, you need to make Vertica aware of it. To do so, use the [CREATE PROCEDURE](#) statement, but review [Creating External Procedures](#) first.

Creating External Procedures

Once you have installed an external procedure, you need to make Vertica aware of it. To do so, use the [CREATE PROCEDURE](#) statement.

Only a superuser can create an external procedure. Initially, only superusers can execute an external procedure. However, a superuser can grant the right to execute a stored procedure to a user on the operating system. (See [GRANT \(Procedure\)](#).)

Once created, a procedure is listed in the `V_CATALOG.USER_PROCEDURES` system table. Users can see only those procedures that they have been granted the privilege to execute.

Example

This example creates a procedure named `helloplanet` for the `helloplanet.sh` external procedure file. This file accepts one `VARCHAR` argument. The sample code is provided in [Requirements for External Procedures](#).

```
=> CREATE PROCEDURE helloplanet(arg1 VARCHAR) AS 'helloplanet.sh' LANGUAGE 'external'
    USER 'dbadmin';
```

This example creates a procedure named `proctest` for the `copy_vertica_database.sh` script. This script copies a database from one cluster to another, and it is included in the server RPM located in the `/opt/vertica/scripts` directory.

```
=> CREATE PROCEDURE proctest(shosts VARCHAR, thosts VARCHAR, dbdir VARCHAR)
  AS 'copy_vertica_database.sh' LANGUAGE 'external' USER 'dbadmin';
```

Overloading External Procedures

You can create multiple external procedures with the same name as long as they have a different signature (accept a different set of arguments). For example, you can overload the `helloplanet` external procedure to also accept an integer value:

```
=> CREATE PROCEDURE helloplanet(arg1 INT) AS 'helloplanet.sh' LANGUAGE 'external'
  USER 'dbadmin';
```

After executing the above statement, your database catalog will have two external procedures named `helloplanet`—one that accepts a `VARCHAR` argument and one that accepts an integer. When you call the external procedure, Vertica determines which procedure to call based on the arguments you passed in the procedure call.

See Also

- [CREATE PROCEDURE](#)
- [GRANT \(Procedure\)](#)

Executing External Procedures

Once you define a procedure through the [CREATE PROCEDURE](#) statement, you can use it as a meta command through a simple `SELECT` statement. Vertica does not support using procedures in more complex statements or in expressions.

The following example runs a procedure named `helloplanet`:

```
=> SELECT helloplanet('earthlings');
helloplanet
-----
          0
(1 row)
```

The following example runs a procedure named `proctest`. This procedure references the `copy_vertica_database.sh` script that copies a database from one cluster to another. It is installed by the server RPM in the `/opt/vertica/scripts` directory.

```
=> SELECT proctest(  
    '-s qa01',  
    '-t rbench1',  
    '-D /scratch_b/qa/PROC_TEST' );
```

Note: External procedures have no direct access to database data. If available, use ODBC or JDBC for this purpose.

Procedures are executed on the initiating node. Vertica runs the procedure by forking and executing the program. Each procedure argument is passed to the executable file as a string. The parent fork process waits until the child process ends.

To stop execution, cancel the process by sending a cancel command (for example, CTRL+C) through the client. If the procedure program exits with an error, an error message with the exit status is returned.

Permissions

To execute an external procedure, the user needs:

- EXECUTE privilege on procedure
- USAGE privilege on schema that contains the procedure

See Also

- [CREATE PROCEDURE](#)
- [External Procedure Privileges](#)

Dropping External Procedures

Only a superuser can drop an external procedure. To drop the definition for an external procedure from Vertica, use the [DROP PROCEDURE](#) statement. Only the reference to the procedure is removed. The external file remains in the <database>/procedures directory on each node in the database.

Note: The definition Vertica uses for a procedure cannot be altered; it can only be dropped.

Example

```
=> DROP PROCEDURE helloplanet(arg1 varchar);
```

See Also

- [DROP PROCEDURE](#)

Using User-Defined SQL Functions

User-Defined SQL Functions let you define and store commonly-used SQL expressions as a function. User-Defined SQL Functions are useful for executing complex queries and combining Vertica built-in functions. You simply call the function name you assigned in your query.

A User-Defined SQL Function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

For syntax and parameters for the commands and system table discussed in this section, see the following topics in the SQL Reference Manual:

- [CREATE FUNCTION](#)
- [ALTER FUNCTION \(UDF or UDT\)](#)
- [DROP FUNCTION](#)
- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)
- [V_CATALOG.USER_FUNCTIONS](#)

Creating User-Defined SQL Functions

A user-defined SQL function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

To create a SQL function, the user must have CREATE privileges on the schema. To use a SQL function, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined function.

This following statement creates a SQL function called `myzeroifnull` that accepts an INTEGER argument and returns an INTEGER result.

```
=> CREATE FUNCTION myzeroifnull(x INT) RETURN INT
AS BEGIN
    RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
END;
```

You can use the new SQL function (`myzeroifnull`) anywhere you use an ordinary SQL expression. For example, create a simple table:

```
=> CREATE TABLE tabwnulls(col1 INT);
=> INSERT INTO tabwnulls VALUES(1);
=> INSERT INTO tabwnulls VALUES(NULL);
=> INSERT INTO tabwnulls VALUES(0);
=> SELECT * FROM tabwnulls;
 a
 ---
 1
 0
 0
(3 rows)
```

Use the `myzeroifnull` function in a `SELECT` statement, where the function calls `col1` from table `tabwnulls`:

```
=> SELECT myzeroifnull(col1) FROM tabwnulls;
 myzeroifnull
 -----
           1
           0
           0
(3 rows)
```

Use the `myzeroifnull` function in the `GROUP BY` clause:

```
=> SELECT COUNT(*) FROM tabwnulls GROUP BY myzeroifnull(col1);
 count
 -----
      2
      1
(2 rows)
```

If you want to change a user-defined SQL function's body, use the `CREATE OR REPLACE` syntax. The following command modifies the `CASE` expression:

```
=> CREATE OR REPLACE FUNCTION myzeroifnull(x INT) RETURN INT
 AS BEGIN
   RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
 END;
```

To see how this information is stored in the Vertica catalog, see [Viewing Information About SQL Functions](#) in this guide.

See Also

- [CREATE FUNCTION \(SQL Functions\)](#)
- [USER_FUNCTIONS](#)

Altering and Dropping User-Defined SQL Functions

Vertica allows multiple functions to share the same name with different argument types. Therefore, if you try to alter or drop a SQL function without specifying the argument data type, the system returns an error message to prevent you from dropping the wrong function:

```
=> DROP FUNCTION myzeroifnull();  
ROLLBACK: Function with specified name and parameters does not exist: myzeroifnull
```

Note: Only a superuser or owner can alter or drop a SQL Function.

Altering a User-Defined SQL Function

The [ALTER FUNCTION \(UDF or UDT\)](#) command lets you assign a new name to a user-defined function, as well as move it to a different schema.

In the previous topic, you created a SQL function called `myzeroifnull`. The following command renames the `myzeroifnull` function to `zerowhennull`:

```
=> ALTER FUNCTION myzeroifnull(x INT) RENAME TO zerowhennull;  
ALTER FUNCTION
```

This next command moves the renamed function into a new schema called `macros`:

```
=> ALTER FUNCTION zerowhennull(x INT) SET SCHEMA macros;  
ALTER FUNCTION
```

Dropping a SQL Function

The [DROP FUNCTION](#) command drops a SQL function from the Vertica catalog.

Like with `ALTER FUNCTION`, you must specify the argument data type or the system returns the following error message:

```
=> DROP FUNCTION zerowhennull();  
ROLLBACK: Function with specified name and parameters does not exist: zerowhennull
```

Specify the argument type:

```
=> DROP FUNCTION macros.zerowhennull(x INT);  
DROP FUNCTION
```

Vertica does not check for dependencies, so if you drop a SQL function where other objects reference it (such as views or other SQL Functions), Vertica returns an error when those objects are used, not when the function is dropped.

Tip: To view a list of all user-defined SQL functions on which you have EXECUTE privileges, (which also returns their argument types), query the [V_CATALOG.USER_FUNCTIONS](#) system table.

See Also

- [ALTER FUNCTION \(UDF or UDT\)](#)
- [DROP FUNCTION](#)

Managing Access to SQL Functions

Before a user can execute a user-defined SQL function, he or she must have USAGE privileges on the schema and EXECUTE privileges on the defined function. Only the superuser or owner can grant/revoke EXECUTE usage on a function.

To grant EXECUTE privileges to user Fred on the `myzeroifnull` function:

```
=> GRANT EXECUTE ON FUNCTION myzeroifnull (x INT) TO Fred;
```

To revoke EXECUTE privileges from user Fred on the `myzeroifnull` function:

```
=> REVOKE EXECUTE ON FUNCTION myzeroifnull (x INT) FROM Fred;
```

See Also

- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)

Viewing Information About User-Defined SQL Functions

You can access information about any User-Defined SQL Functions on which you have EXECUTE privileges. This information is available in the system table [V_CATALOG.USER_FUNCTIONS](#) and from the vsql meta-command `\df`.

To view all of the User-Defined SQL Functions on which you have EXECUTE privileges, query the [USER_FUNCTIONS](#) table:

```
=> SELECT * FROM USER_FUNCTIONS;
-[ RECORD 1 ]-----+-----
schema_name         | public
function_name       | myzeroifnull
function_return_type | Integer
function_argument_type | x Integer
function_definition | RETURN CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END
volatility           | immutable
is_strict            | f
```

If you want to change a User-Defined SQL Function's body, use the CREATE OR REPLACE syntax. The following command modifies the CASE expression:

```
=> CREATE OR REPLACE FUNCTION myzeroifnull(x INT) RETURN INT
AS BEGIN
  RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
END;
```

Now when you query the [USER_FUNCTIONS](#) table, you can see the changes in the `function_definition` column:

```
=> SELECT * FROM USER_FUNCTIONS;
-[ RECORD 1 ]-----+-----
schema_name         | public
function_name       | myzeroifnull
function_return_type | Integer
function_argument_type | x Integer
function_definition | RETURN CASE WHEN (x IS NULL) THEN 0 ELSE x END
volatility           | immutable
is_strict            | f
```

If you use CREATE OR REPLACE syntax to change only the argument name or argument type (or both), the system maintains both versions of the function. For example, the following command tells the function to accept and return a numeric data type instead of an integer for the `myzeroifnull` function:

```
=> CREATE OR REPLACE FUNCTION myzeroifnull(z NUMERIC) RETURN NUMERIC
AS BEGIN
  RETURN (CASE WHEN (z IS NULL) THEN 0 ELSE z END);
END;
```

Now query the `USER_FUNCTIONS` table, and you can see the second instance of `myzeroifnull` in Record 2, as well as the changes in the `function_return_type`, `function_argument_type`, and `function_definition` columns.

Note: Record 1 still holds the original definition for the `myzeroifnull` function:

```
=> SELECT * FROM USER_FUNCTIONS;
-[ RECORD 1 ]-----+-----
schema_name         | public
function_name       | myzeroifnull
function_return_type | Integer
function_argument_type | x Integer
function_definition | RETURN CASE WHEN (x IS NULL) THEN 0 ELSE x END
volatility           | immutable
is_strict            | f
-[ RECORD 2 ]-----+-----
schema_name         | public
function_name       | myzeroifnull
function_return_type | Numeric
function_argument_type | z Numeric
function_definition | RETURN (CASE WHEN (z IS NULL) THEN (0) ELSE z END)::numeric
volatility           | immutable
is_strict            | f
```

Because Vertica allows functions to share the same name with different argument types, you must specify the argument type when you [alter](#) or [drop](#) a function. If you do not, the system returns an error message:

```
=> DROP FUNCTION myzeroifnull();
ROLLBACK: Function with specified name and parameters does not exist: myzeroifnull
```

See Also

- [USER_FUNCTIONS](#)

Migrating Built-In SQL Functions

If you have built-in SQL functions from another RDBMS that do not map to an Vertica-supported function, you can migrate them into your Vertica database by using a user-defined SQL function.

The example scripts below show how to create user-defined functions for the following DB2 built-in functions:

- UCASE()
- LCASE()
- LOCATE()
- POSSTR()

UCASE()

This script creates a user-defined SQL function for the UCASE() function:

```
=> CREATE OR REPLACE FUNCTION UCASE (x VARCHAR)
  RETURN VARCHAR
  AS BEGIN
  RETURN UPPER(x);
  END;
```

LCASE()

This script creates a user-defined SQL function for the LCASE() function:

```
=> CREATE OR REPLACE FUNCTION LCASE (x VARCHAR)
  RETURN VARCHAR
  AS BEGIN
  RETURN LOWER(x);
  END;
```

LOCATE()

This script creates a user-defined SQL function for the LOCATE() function:

```
=> CREATE OR REPLACE FUNCTION LOCATE(a VARCHAR, b VARCHAR)
  RETURN INT
  AS BEGIN
  RETURN POSITION(a IN b);
  END;
```

POSSTR()

This script creates a user-defined SQL function for the POSSTR() function:

```
=> CREATE OR REPLACE FUNCTION POSSTR(a VARCHAR, b VARCHAR)
    RETURN INT
    AS BEGIN
    RETURN POSITION(b IN a);
    END;
```

Using User-Defined Extensions

A User-Defined Extension (abbreviated as UDx) is a component that adds new abilities to the Vertica Analytics Platform. UDxs provide features such as new types of data analysis and the ability to parse and load new types of data.

This section provides an overview of how to install and use a UDx. If you are using a UDx developed by a third party, you should consult any documentation that accompanies it for detailed installation and usage instructions.

Loading UDxs

User-Defined Extensions (UDxs) are contained in libraries. A library can contain multiple UDxs. To add UDxs to Vertica you must:

1. Load the library (once per library).
2. Declare each UDx (once per UDx).

If you are using UDxs written in Java, you must also set up a Java runtime environment. See [Installing Java on Vertica Hosts](#).

Loading Libraries

To deploy a library to your Vertica database:

1. Copy the UDx shared library file (.so), Python file, or JAR that contains your function to a node on your Vertica cluster. You do not need to copy it to every node.
2. Connect to the node where you copied the library (for example, using vsqll).
3. Add your library to the database catalog using the [CREATE LIBRARY](#) statement.

```
=> CREATE LIBRARY libname AS '/path_to_lib/filename'  
LANGUAGE 'language';
```

libname is the name you want to use to reference the library. *path_to_lib/filename* is the fully-qualified path to the library or JAR file you copied to the host. *language* is the implementation language.

For example, if you created a JAR file named `TokenizeStringLib.jar` and copied it to the `dbadmin` account's home directory, you would use this command to load the library:

```
=> CREATE LIBRARY tokenizeLib AS '/home/dbadmin/TokenizeStringLib.jar'  
LANGUAGE 'Java';
```

You can load any number of libraries into Vertica.

Declaring Individual UDxs

Once the library is loaded, define individual UDxs using SQL statements such as [CREATE FUNCTION](#) and [CREATE SOURCE](#). These statements assign SQL function names to the extension classes in the library. They add the UDx to the database catalog and remain available after a database restart.

The statement you use depends on the type of UDx you are declaring, as shown in the following table.

UDx Type	SQL Statement
Aggregate Function (UDAF)	CREATE AGGREGATE FUNCTION
Analytic Function (UDAnF)	CREATE ANALYTIC FUNCTION
Scalar Function (UDSF)	CREATE FUNCTION (UDF)
Transform Function (UDTF)	CREATE TRANSFORM FUNCTION
Load (UDL): Source	CREATE SOURCE
Load (UDL): Filter	CREATE FILTER
Load (UDL): Parser	CREATE PARSER

After you add the UDx to the database, you can use your extension within SQL statements. The database superuser can grant access privileges to the UDx for users. See [GRANT \(User Defined Extension\)](#) in the SQL Reference Manual for details.

When you call a UDx, Vertica creates an instance of the UDx class on each node in the cluster and provides it with the data it needs to process.

Installing Java on Vertica Hosts

If you are using UDxs written in Java, follow the instructions in this section.

You must install a Java Virtual Machine (JVM) on every host in your cluster in order for Vertica to be able to execute your Java UDxs.

Installing Java on your Vertica cluster is a two-step process:

1. Install a Java runtime on all of the hosts in your cluster.
2. Set the `JavaBinaryForUDx` configuration parameter to tell Vertica the location of the Java executable.

Installing a Java Runtime

For Java-based features, Vertica requires a 64-bit Java 6 (Java version 1.6) or later Java runtime. Vertica supports runtimes from either Oracle or [OpenJDK](#). You can choose to install either the Java Runtime Environment (JRE) or Java Development Kit (JDK), since the JDK also includes the JRE.

Many Linux distributions include a package for the OpenJDK runtime. See your Linux distribution's documentation for information about installing and configuring OpenJDK.

To install the Oracle Java runtime, see the [Java Standard Edition \(SE\) Download Page](#). You usually run the installation package as root in order to install it. See the download page for instructions.

Once you have installed a JVM on each host, ensure that the `java` command is in the search path and calls the correct JVM by running the command:

```
$ java -version
```

This command should print something similar to:

```
java version "1.8.0_102"  
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)  
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
```

Note: Any previously installed Java VM on your hosts may interfere with a newly installed Java runtime. See your Linux distribution's documentation for instructions on configuring which JVM is the default. Unless absolutely required, you should uninstall any incompatible

version of Java before installing the Java 6 or Java 7 runtime.

Setting the JavaBinaryForUDx Configuration Parameter

The JavaBinaryForUDx configuration parameter tells Vertica where to look for the JRE to execute Java UDxs. After you have installed the JRE on all of the nodes in your cluster, set this parameter to the absolute path of the Java executable. You can use the symbolic link that some Java installers create (for example `/usr/bin/java`). If the Java executable is in your shell search path, you can get the path of the Java executable by running the following command from the Linux command line shell:

```
$ which java
/usr/bin/java
```

If the `java` command is not in the shell search path, use the path to the Java executable in the directory where you installed the JRE. Suppose you installed the JRE in `/usr/java/default` (which is where the installation package supplied by Oracle installs the Java 1.6 JRE). In this case the Java executable is `/usr/java/default/bin/java`.

You set the configuration parameter by executing the following statement as a database superuser:

```
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
```

See [ALTER DATABASE](#) for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the [CONFIGURATION_PARAMETERS](#) system table:

```
=> \x
Expanded display is on.
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'JavaBinaryForUDx';
-[ RECORD 1 ]-----+-----
node_name          | ALL
parameter_name     | JavaBinaryForUDx
current_value      | /usr/bin/java
default_value      |
change_under_support_guidance | f
change_requires_restart | f
description        | Path to the java binary for executing UDx written in Java
```

Once you have set the configuration parameter, Vertica can find the Java executable on each node in your cluster.

Note: Since the location of the Java executable is set by a single configuration parameter for the entire cluster, you must ensure that the Java executable is installed in the same

path on all of the hosts in the cluster.

Notes on Individual UDX Types

Some UDX types have special considerations or restrictions.

Analytic Functions

UDAnFs do not support [framing windows using ROWS](#).

As with Vertica's built-in analytic functions, UDAnFs cannot be used with [Pattern Matching Functions](#).

Transform Functions

A query that includes a UDTF cannot contain:

- Any statements other than the [SELECT](#) statement containing the call to the UDTF and a PARTITION BY expression
- Any other [analytic function](#)
- A call to another UDTF
- A [TIMESERIES](#) clause
- A [pattern matching](#) clause
- A [gap filling and interpolation](#) clause

Load

Installing an untrusted UDL function can compromise the security of the server. UDX's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations.

Superusers must not grant access to UDXs to untrusted users.

You cannot ALTER UDL functions.

Fenced Mode

User-Defined Extensions (UDxs) written in the C++ programming language have the option of running in unfenced mode, which means running directly within the Vertica process. Since they run within Vertica, unfenced UDxs have little overhead, and can perform almost as fast as Vertica's own built-in functions. However, since they run within Vertica directly, any bugs in their code (memory leaks, for example) can destabilize the main Vertica process that can bring one or more database nodes down.

You can instead opt to run most C++ UDxs in fenced mode, which runs the UDxs code outside of the main Vertica process in a separate zygote process. UDx code that crashes while running in fenced mode does not impact the core Vertica process. There is a small performance impact when running UDx code in fenced mode. On average, using fenced mode adds about 10% more time to execution compared to unfenced mode.

Fenced mode is currently available for all C++ UDxs with the exception of User-Defined Aggregates. All UDxs developed in the Python, R, and Java programming languages must run in fenced mode, since the Python, R, and Java runtimes cannot be directly run within the Vertica process.

Using fenced mode does not affect the development of your UDx. Fenced mode is enabled by default for UDx's that support fenced mode. The CREATE FUNCTION command can optionally be issued with the NOT FENCED modifier to disable fenced mode for the function. Also, you can enable or disable fenced mode on any fenced mode-supported C++ UDx by using the ALTER FUNCTION command.

About the Zygote Process

The Vertica zygote process starts when Vertica starts. Each node has a single zygote process. Side processes are created "on demand". The zygote listens for requests and spawns a UDx side session that runs the UDx in fenced mode when a UDx is called by the user.

About Fenced Mode Logging:

UDx code that runs in fenced mode is logged in the `UDxZygote.log` and is stored in the `UDxLogs` directory in the catalog directory of Vertica. Log entries for the side process are denoted by the UDx language (for example, C++), node, and zygote process ID, and the `UDxSideProcess ID`.

For example, for the following processes...

```
=> SELECT * FROM UDX_FENCED_PROCESSES;
 node_name | process_type | session_id | pid | port | status
-----+-----+-----+-----+-----+-----
 v_vmart_node0001 | UDXZygoteProcess | | 27468 | 51900 | UP
 v_vmart_node0001 | UDXSideProcess | localhost.localdoma-27465:0x800b | 5677 | 44123 | UP
```

... the corresponding log file displays:

```
2016-05-16 11:24:43.990 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7fd0 UDX side process
started
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7fd0 Finished setting up signal
handlers.
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7fd0 My port: 44123
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7fd0 My address: 0.0.0.0
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7fd0 Vertica port: 51900
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7fd0 Vertica address: 127.0.0.1
11:25:19.749 [C++-localhost.localdoma-27465:0x800b-5677] 0x41837940 Setting memory resource limit
to -1
11:30:11.523 [C++-localhost.localdoma-27465:0x800b-5677] 0x41837940 Exiting UDX side process
```

The last line indicates that the side process was killed. In this case it was killed when the user session (vsq!) closed.

About Fenced Mode Configuration Parameters

Fenced mode supports three [configuration parameters](#):

- `FencedUDxMemoryLimitMB` - The maximum memory size, in MB, to use for Fenced Mode processes. The default is -1 (no limit). The side process is killed if this limit is exceeded.
- `ForceUDxFencedMode` - When set to 1, force all UDX's that support fenced mode to run in fenced mode even if their definition specified NOT FENCED. The default is 0 (disabled).
- `UDxFencedBlockTimeout` - The maximum time, in seconds, that the Vertica server waits for a UDX to return before aborting with ERROR 3399. The default is 60.

See Also

- [CREATE LIBRARY](#)
- [CREATE FUNCTION](#)
- [CREATE TRANSFORM FUNCTION](#)

- [CREATE ANALYTIC FUNCTION](#)
- [ALTER FUNCTION \(UDF or UDT\)](#)
- [UDX_FENCED_PROCESSES](#)

Updating UDX Libraries

There are two cases where you need to update libraries that you have already deployed:

- When you have upgraded Vertica to a new version that contains changes to the SDK API. For your libraries to work with the new server version, you need to recompile them with new version of the SDK. See [UDx Library Compatibility with New Server Versions](#) for more information.
- When you have made changes to your UDXs and you want to deploy these changes. Before updating your UDX library, you need to determine if you have changed the signature of any of the functions contained in the library. If you have, you need to drop the functions from the Vertica catalog before you update the library.

UDx Library Compatibility with New Server Versions

The Vertica SDK defines an application programming interface (API) that UDXs use to interact with the database. When a developer compiles his or her UDX code, it is linked to the SDK code to form a library. This library is only compatible with Vertica servers that support the version of the SDK API used to compile the code. The library and servers that share the same API version are compatible on a binary level (referred to as "binary compatible").

The Vertica server returns an error message if you attempt to load a library that is not binary compatible with it. Similarly, if you upgrade your Vertica server to a version that supports a new SDK API, any existing UDX that relies on newly-incompatible libraries returns an error messages when you call it:

```
ERROR 2858: Could not find function definition
HINT:
This usually happens due to missing or corrupt libraries, libraries built
with the wrong SDK version, or due to a concurrent session dropping the library
or function. Try recreating the library and function
```

To resolve this issue, you must install UDX libraries that have been recompiled with correct version of the SDK.

New versions of the Vertica server do not always change the SDK API version. The SDK API version changes whenever Micro Focus changes the components that make up the SDK. If the SDK API does not change in a new version of the server, then the old libraries remain compatible with the new server.

The SDK API almost always changes between major releases of Vertica (for example between versions 6.0 and 6.1), as Micro Focus expands the SDK's features in each major release. In addition, Micro Focus reserves the right to change the SDK API in the first service pack release (for example between (from version 7.0.0 to 7.0.1) to address any issues found in the initial release of a major version. After the first service pack, the SDK API is frozen and will not change for rest of the service packs for that major release (for example, 7.0.2, 7.0.3, and so forth).

These policies mean that:

- You need to update UDx libraries when you upgrade between major versions. For example, if you upgrade from version 7.0.1 to 7.1.0 you need to update your UDx libraries.
- You may need to update the UDx libraries if you upgrade from the first version of a major release to a later service pack release. For example, if you upgrade the server from version 7.1.0 to 7.1.2, you may need to update your UDx libraries. Version 7.1.1 (the first service pack release) may have SDK changes that make your libraries compiled with the 7.1.0 SDK incompatible with the Vertica 7.1.2 server.
- You do not need to update your UDx libraries when upgrading from one service pack to another. For example, if you upgrade from version 7.1.1 to 7.1.2, you do not need to update your UDx libraries. The SDK API version is the same between these versions and any later 7.1.x service pack releases.

Note: Since the R language is interpreted, a UDx written in R is not linked to the Vertica SDK. There is no binary compatibility that has to be maintained from one version to another. Therefore, changes to the Vertica SDK between Vertica versions do not make your R-based UDx libraries incompatible with a new server version. An R-based UDx only becomes incompatible if the APIs used in the SDK actually change. For example, if the number of arguments to an API call changes, the UDx written in R has to be changed to use the new number of arguments.

Pre-Upgrade Steps

Before upgrading your Vertica server, consider whether you have any UDx libraries that may be incompatible with the new version. Consult the release notes of the new server version to determine whether the SDK API has changed between the version of Vertica server you currently have installed and the new version. As mentioned previously, only upgrades from a previous major version or from the initial release of a major version to a service pack release can cause your currently-loaded UDx libraries to become incompatible with the server.

Any UDx libraries that are incompatible with the new version of the Vertica server must be recompiled. If you got the UDx library from a third party (such as through the Vertica

Marketplace), you need to see if a new version has been released. If so, then download the UDX library and deploy it after you have upgraded the server (see [Deploying A New Version of Your UDX Library](#)).

If you developed the UDX yourself (or if whoever developed it supplied the UDX's source code) you must:

1. Recompile your UDX library using the new version of the Vertica SDK. See [Compiling Your C++ Library](#) or [Compiling and Packaging a Java Library](#) for more information.
2. Deploy the new version of your library. See [Deploying A New Version of Your UDX Library](#).

Determining If a UDX Signature Has Changed

You need to be careful when making changes to UDX libraries that contain functions you have already deployed in your Vertica database. When you deploy a new version of your UDX library, Vertica does not ensure that the signatures of the functions that are defined in the library match the signature of the function that is already defined in the Vertica catalog. If you have changed the signature of a UDX in the library then update the library in the Vertica database, calls to the altered UDX will produce errors.

Making any of the following changes to a UDX alters its signature:

- Changing the number of arguments accepted or the data type of any argument accepted by your function (not including polymorphic functions).
- Changing the number or data types of any return values or output columns.
- Changing the name of the factory class that Vertica uses to create an instance of your function code.
- Changing the null handling or volatility behavior of your function.
- Removed the function's factory class from the library completely.

The following changes do not alter the signature of your function, and do not require you to drop the function before updating the library:

- Changing the number or type of arguments handled by a polymorphic function. Vertica does not process the arguments the user passes to a polymorphic function.
- Changing the name, data type, or number of parameters accepted by your function. The parameters your function accepts are not determined by the function signature. Instead,

Vertica passes all of the parameters the user included in the function call, and your function processes them at runtime. See [UDx Parameters](#) for more information about parameters.

- Changing any of the internal processing performed by your function.
- Adding new UDxs to the library.

After you drop any functions whose signatures have changed, you load the new library file, then re-create your altered functions. If you have not made any changes to the signature of your UDxs, you can just update the library file in your Vertica database without having to drop or alter your function definitions. As long as the UDx definitions in the Vertica catalog match the signatures of the functions in your library, function calls will work transparently after you have updated the library. See [Deploying A New Version of Your UDx Library](#).

Deploying A New Version of Your UDx Library

You need to deploy a new version of your UDx library if:

- You have made changes to the library that you now want to roll out to your Vertica database.
- You have upgraded your Vertica to a new version whose SDK is incompatible with the previous version.

The process of deploying a new version of your library is similar to deploying it initially.

1. If you are deploying a UDx library developed in C++ or Java, you must compile it with the current version of the Vertica SDK.
2. Copy your UDx's library file (a `.so` file for libraries developed in C++, a `.py` file for libraries developed in Python, or a `.jar` file for libraries developed in Java) or R source file to a host in your Vertica database.
3. Connect to the host using `vsq`.
4. If you have changed the signature of any of the UDxs in the shared library, you must drop them using `DROP` statements such as [DROP FUNCTION](#) or [DROP SOURCE](#). If you are unsure whether any of the signatures of your functions have changed, see [Determining If a UDx Signature Has Changed](#).

Note: If all of the UDX signatures in your library have changed, you may find it more convenient to drop the library using the [DROP LIBRARY](#) statement with the `CASCADE` option to drop the library and all of the functions and loaders that reference it. Dropping the library can save you the time it would take to drop each UDX individually. You can then reload the library and recreate all of the extensions using the same process you used to deploy the library in the first place. See [CREATE LIBRARY](#) in the SQL Reference Manual.

5. Use the [ALTER LIBRARY](#) statement to update the UDX library definition with the file you copied in step 1. For example, if you want to update the library named `ScalarFunctions` with a file named `ScalarFunctions-2.0.so` in the `dbadmin` user's home directory, you could use the command:

```
=> ALTER LIBRARY ScalarFunctions AS '/home/dbadmin/ScalarFunctions-2.0.so';
```

Once you have updated the UDX library definition to use the new version of your shared library, the UDXs that are defined using classes in your UDX library begin using the new shared library file without any further changes.

6. If you had to drop any functions in step 4, recreate them using the new signature defined by the factory classes in your library. See [CREATE FUNCTION Statements](#) in the SQL Reference Manual.

Listing the UDxs Contained in a Library

Once a library has been loaded using the [CREATE LIBRARY](#) statement, you can find the UDxs and UDLs it contains by querying the [USER_LIBRARY_MANIFEST](#) system table:

```
=> CREATE LIBRARY ScalarFunctions AS '/home/dbadmin/ScalarFunctions.so';
CREATE LIBRARY
=> \x
Expanded display is on.
=> SELECT * FROM USER_LIBRARY_MANIFEST WHERE lib_name = 'ScalarFunctions';
-[ RECORD 1 ]-----
schema_name | public
lib_name    | ScalarFunctions
lib_oid     | 45035996273792402
obj_name    | RemoveSpaceFactory
obj_type    | Scalar Function
arg_types   | Varchar
return_type | Varchar
-[ RECORD 2 ]-----
schema_name | public
lib_name    | ScalarFunctions
lib_oid     | 45035996273792402
obj_name    | Div2intsInfo
obj_type    | Scalar Function
arg_types   | Integer, Integer
return_type | Integer
-[ RECORD 3 ]-----
schema_name | public
lib_name    | ScalarFunctions
lib_oid     | 45035996273792402
obj_name    | Add2intsInfo
obj_type    | Scalar Function
arg_types   | Integer, Integer
return_type | Integer
```

The `obj_name` column lists the factory classes contained in the library. These are the names you use to define UDxs and UDLs in the database catalog using statements such as [CREATE FUNCTION](#) and [CREATE SOURCE](#).

Using Wildcards In Your UDx

Vertica supports wildcard `*` characters in the place of column names in user-defined functions.

You can use wildcards when:

- Your query contains a table in the FROM clause
- You are using a Vertica-supported development language
- Your UDX is running in fenced or unfenced mode

Supported SQL Statements

The following SQL statements can accept wildcards:

- DELETE
- INSERT
- SELECT
- UPDATE

Unsupported Configurations

The following situations do not support wildcards:

- You cannot pass a wildcard in the OVER clause of a query
- You cannot use a wildcard with a DROP statement
- You cannot use wildcards with any other arguments

Examples

These examples show wildcards and user-defined functions in a range of data manipulation operations.

DELETE statements:

```
=> DELETE FROM tablename WHERE udf(tablename.*) = 5;
```

INSERT statements:

```
=> INSERT INTO table1 SELECT udf(*) FROM table2;
```

SELECT statements:

```
=> SELECT udf(*) FROM tablename;  
=> SELECT udf(tablename.*) FROM tablename;  
=> SELECT udf(f.*) FROM table f;  
=> SELECT udf(*) FROM table1,table2;  
=> SELECT udf1( udf2(*) ) FROM table1,table2;  
=> SELECT udf( db.schema.table.*) FROM tablename;  
=> SELECT udf(sub.*) FROM (select col1, col2 FROM table) sub;  
=> SELECT x FROM tablename WHERE udf(*) = y;  
=> WITH sub as (SELECT * FROM tablename) select x, udf(*) FROM sub;  
=> SELECT udf( * using parameters x=1) FROM tablename;  
=> SELECT udf(table1.*, table2.col2) FROM table1,table2;
```

UPDATE statements:

```
=> UPDATE tablename set col1 = 4 FROM tablename WHERE udf(*) = 3;
```

Developing User-Defined Extensions (UDxs)

User-Defined Extensions (UDxs) are functions contained in external shared libraries that are developed in C++, Python, Java, or R using the Vertica SDK. The external libraries are defined in the Vertica catalog using the [CREATE LIBRARY](#) statement. They are best suited for analytic operations that are difficult to perform in SQL, or need to be performed frequently enough that their speed is a major concern.

The primary strengths of UDxs are:

- They can be used anywhere an internal function can be used.
- They take full advantage of Vertica's distributed computing features. The extensions usually execute in parallel on each node in the cluster.
- Vertica handles the distribution of the UDx library to the individual nodes. You only need to copy the library to the initiator node.
- All of the complicated aspects of developing a distributed piece of analytic code are handled for you by Vertica. Your main programming task is to read in data, process it, and then write it out using the Vertica SDK APIs.

There are a few things to keep in mind about developing UDxs:

- UDxs can be developed in the programming languages C++, Python, Java, and R. (Not all UDx types support all languages.)
- UDxs written in Java always run in [Fenced Mode](#), because the Java Virtual Machine that executes Java programs cannot run directly within the Vertica process.
- UDxs written in Python and R always run in [Fenced Mode](#)
- UDxs developed in C++ have the option of running in unfenced mode, which means they load and run directly in the Vertica database process. This option provides the lowest overhead and highest speed. However, any bugs in the UDx's code can cause database instability. You must thoroughly test any UDxs you intend to run in unfenced mode before deploying them in a live environment. Consider whether the performance boost of running a C++ UDx unfenced is worth the potential database instability that a buggy UDx can cause.

- Because a UDX runs on the Vertica cluster, it can take processor time and memory away from the database processes. A UDX that consumes large amounts of computing resources can negatively impact database performance.

Structure

Each UDX type consists of two classes. The main class does the primary work (a transformation, an aggregation, and so on). The class usually has at least three methods: one to set up, one to tear down (release reserved resources), and one to do the actual work. Sometimes additional methods are defined.

The main processing method receives an instance of the `ServerInterface` class as an argument. This object is used by the underlying Vertica SDK code to make calls back into the Vertica process, for example to allocate memory. You can use this class to write to the server log during UDX execution.

The second class is a singleton factory. It defines one method that produces instances of the first class, and might define other methods to manage parameters.

When implementing a UDX you must subclass both classes.

Conventions

The C++, Python, and Java APIs are nearly identical. Where possible, this documentation describes these interfaces without respect to language. Documentation specific to C++, Python, or Java is covered in language-specific sections.

Because some documentation is language-independent, it is not always possible to use ideal, language-based terminology. This documentation uses the term "method" to refer to a Java method or a C++ member function.

See Also

[Loading UDXs](#)

Setting Up a Development Environment

Before you start developing your UDX, you need to configure your development environment. You can choose to develop your UDX on a node in a development Vertica database (not in a production environment) or on another machine.

To test, you need access to a (non-production) Vertica database. You can install a single-node Vertica database on your development machine for easier development.

You should develop your UDX code on the same Linux platform that you use on your Vertica database cluster. This will ensure that your UDX library is compatible with the Vertica version deployed on your cluster.

The following sections describe language-specific requirements.

C++ Requirements

At a minimum, you need to install the following on your development machine:

- [g++](#) and its associated tool chain such as `ld`. (**Note:** some Linux distributions package `g++` separately from `gcc`.)
- A copy of the Vertica SDK. See [Setting Up the C++ SDK](#) for details.

Note: The Vertica binaries are compiled using the default version of `g++` installed on the supported Linux platforms. While other versions of `g++` (or even entirely different compilers) may produce compatible libraries, only the platform's default `g++` version is supported for compiling UDXs.

While not required, the following additional software packages can ease development:

- `make`, or some other build-management tool.
- `gdb`, or some other debugger.
- `Valgrind`, or similar tools that detect memory leaks.

If you want to use any third-party libraries (for example, statistical analysis libraries), you need to install them on your development machine. If you do not statically link these libraries into your UDX library, you also have to install them on every node in the cluster. See [Compiling Your C++ Library](#) for details.

Java Requirements

At a minimum, you need to install the following on your development machine:

- The Java Development Kit (JDK) version that matches the Java version you have installed on your database hosts (see [Installing Java on Vertica Hosts](#)).
- A copy of the Vertica SDK. See [Setting Up the Java SDK](#) for details.

While not required, make or some other build-management tool can ease development.

Python Requirements

Vertica does not require any additional files or packages. You can develop your Python UDx on any system with a text editor.

Important: When you begin UDx development, make sure that your code runs using Python 3.5.1. because you cannot change the version used by the Vertica Python interpreter.

When Vertica calls your UDx, it starts a side process that manages the interaction between the server and the Python interpreter.

R Requirements

Vertica does not require any additional files or packages. You can develop your R UDx on any system with a text editor.

Downloading and Running UDx Example Code

You can download all of the examples shown in this documentation, and many more, from the Vertica [GitHub repository](#). This repository includes examples of all types of UDxs.

You can download the examples in either of two ways:

- Download the ZIP file. Extract the contents of the file into a directory.
- Clone the repository. Using a terminal window, run the following command:

```
$ git clone https://github.com/vertica/UDx-Examples.git
```

The repository includes a makefile that you can use to compile the C++ and Java examples. It also includes .sql files that load and use the examples. See the README file for instructions on compiling and running the examples.

Running the examples not only helps you understand how a UDx works, but also helps you ensure your development environment is properly set up to compile UDx libraries.

See Also

- [Aggregate Functions \(UDAFs\)](#)
- [Analytic Functions \(UDAnFs\)](#)
- [Scalar Functions \(UDSFs\)](#)
- [Transform Functions \(UDTFs\)](#)
- [Load \(UDLs\)](#)

Types of UDxs

Vertica supports five types of user-defined extensions:

- User-Defined Aggregate Functions (UDAF) allow you to create custom [Aggregate Functions](#) specific to your needs. They read one column of data, and return one output column. UDAFs can be developed in C++. Developing UDAFs is described in [Aggregate Functions \(UDAFs\)](#).
- User-Defined Analytic Functions (UDAnF) are similar to UDSFs, in that they read a row of data and return a single row. However, the function can read input rows independently of outputting rows, so that the output values can be calculated over several input rows.

UDAnFs can be developed in C++ and Java. Developing UDAnFs is described in [Analytic Functions \(UDAnFs\)](#).

- User-Defined Scalar Functions (UDSFs) take in a single row of data and return a single value. These functions can be used anywhere a native function can be used, except CREATE TABLE BY PARTITION and SEGMENTED BY expressions. UDSFs can be developed in C++, Python, Java, and R. Developing UDSFs is described in [Scalar Functions \(UDSFs\)](#) and [Developing UDSFs and UDTFs in R](#).
- User-Defined Transform Functions (UDTFs) operate on table segments and return zero or more rows of data. The data they return can be an entirely new table, unrelated to the schema of the input table, including having its own ordering and segmentation expressions. They can only be used in the SELECT list of a query. UDTFs can be developed in C++, Java, and R. Developing UDTFs is described in [Transform Functions \(UDTFs\)](#) and [Developing UDSFs and UDTFs in R](#).

To optimize query performance, you can use live aggregate projections to pre-aggregate the data that a UDTF returns. For more information, see [Pre-Aggregating UDTF Results](#).

- User-Defined Load allows you to create custom sources, filters, and parsers to load data. These extensions are used in the [COPY](#) statement. UDLs can be developed in C++ and Java. Developing UDLs is described in [Load \(UDLs\)](#).

While each UDx type has a unique base class, developing them is similar in many ways. Different UDx types can also share the same library.

Handling Different Numbers and Types of Arguments

Usually, your UDxs accept a set number of arguments that are a specific data type (called its signature). You can create UDxs that handle multiple signatures, or even accept all arguments supplied to them by the user, using either of these techniques:

- Overloading your UDx by assigning the same SQL function name to multiple factory classes, each of which defines a unique function signature. When a user uses the function name in a query, Vertica tries to match the signature of the function call to the signatures declared by the factory's `getPrototype` method. This is the best technique to use if your UDx needs to accept a few different signatures (for example, accepting two required and one optional argument).
- Creating a polymorphic UDx by using the special "Any" argument type that tells Vertica to send all arguments that the user supplies to your function. Your UDx decides whether it can handle the arguments or not.

Overloading Your UDx

You may want your UDx to accept several different signatures (sets of arguments). For example, you might want your UDx to accept:

- One or more optional arguments.
- One or more arguments that can be one of several data types.
- Completely distinct signatures (either all INTEGER or all VARCHAR, for example).

You can create a function with this behavior by creating several factory classes, each of which accepts a different signature (the number and data types of arguments). You can then associate a single SQL function name with all of them. You can use the same SQL function name to refer to multiple factory classes as long as the signature defined by each factory is unique. When a user calls your UDx, Vertica matches the number and types of arguments supplied by the user to the arguments accepted by each of your function's factory classes. If one matches, Vertica uses it to instantiate a function class to process the data.

Multiple factory classes can instantiate the same function class, so you can re-use one function class that is able to process multiple sets of arguments and then create factory classes for each of the function signatures. You can also create multiple function classes if you want.

See the [C++ Example: Overloading Your UDX](#) and [Java Example: Overloading Your UDX](#) examples.

C++ Example: Overloading Your UDX

The following example code demonstrates creating a User-Defined Scalar Function (UDSF) that adds two or three integers together. The `Add2or3ints` class is prepared to handle two or three arguments. The `processBlock()` function checks the number of arguments that have been passed to it, and adds all two or three of them together. It also exits with an error message if it has been called with less than 2 or more than 3 arguments. In theory, this should never happen, since Vertica only calls the UDSF if the user's function call matches a signature on one of the factory classes you create for your function. In practice, it is a good idea to perform this sanity checking, in case your (or someone else's) factory class inaccurately reports a set of arguments your function class cannot handle.

```
#include "Vertica.h"
using namespace Vertica;
using namespace std;
// a ScalarFunction that accepts two or three
// integers and adds them together.
class Add2or3ints : public Vertica::ScalarFunction
{
public:
    virtual void processBlock(Vertica::ServerInterface &srvInterface,
                             Vertica::BlockReader &arg_reader,
                             Vertica::BlockWriter &res_writer)
    {
        const size_t numCols = arg_reader.getNumCols();

        // Ensure that only two or three parameters are passed in
        if ( numCols < 2 || numCols > 3)
            vt_report_error(0, "Function only accept 2 or 3 arguments, "
                           "but %zu provided", arg_reader.getNumCols());

        // Add two integers together
        do {
            const vint a = arg_reader.getIntRef(0);
            const vint b = arg_reader.getIntRef(1);
            vint c = 0;
            // Check for third argument, add it in if it exists.
            if (numCols == 3)
                c = arg_reader.getIntRef(2);
            res_writer.setInt(a+b+c);
            res_writer.next();
        } while (arg_reader.next());
    }
};
// This factory accepts function calls with two integer arguments.
```

```
class Add2intsFactory : public Vertica::ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface
        &srvInterface)
    { return vt_createFuncObj(srvInterface.allocator, Add2or3ints); }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
        Vertica::ColumnTypes &argTypes,
        Vertica::ColumnTypes &returnType)
    { // Accept 2 integer values
      argTypes.addInt();
      argTypes.addInt();
      returnType.addInt();
    }
};
RegisterFactory(Add2intsFactory);
// This factory defines a function that accepts 3 ints.
class Add3intsFactory : public Vertica::ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface
        &srvInterface)
    { return vt_createFuncObj(srvInterface.allocator, Add2or3ints); }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
        Vertica::ColumnTypes &argTypes,
        Vertica::ColumnTypes &returnType)
    { // accept 3 integer values
      argTypes.addInt();
      argTypes.addInt();
      argTypes.addInt();
      returnType.addInt();
    }
};
RegisterFactory(Add3intsFactory);
```

The example has two `ScalarFunctionFactory` classes, one for each signature that the function accepts (two integers and three integers). There is nothing unusual about these factory classes, except that their implementation of `ScalarFunctionFactory::createScalarFunction()` both create `Add2or3ints` objects.

The final step is to bind the same SQL function name to both factory classes. You can assign multiple factories to the same SQL function, as long as the signatures defined by each factory's `getPrototype()` implementation are different.

```
=> CREATE LIBRARY add2or3IntsLib AS '/home/dbadmin/Add2or3Ints.so';
CREATE LIBRARY
=> CREATE FUNCTION add2or3Ints as NAME 'Add2intsFactory' LIBRARY add2or3IntsLib FENCED;
CREATE FUNCTION
=> CREATE FUNCTION add2or3Ints as NAME 'Add3intsFactory' LIBRARY add2or3IntsLib FENCED;
CREATE FUNCTION
=> SELECT add2or3Ints(1,2);
    add2or3Ints
    -----
           3
(1 row)
=> SELECT add2or3Ints(1,2,4);
```

```
add2or3Ints
-----
          7
(1 row)
=> SELECT add2or3Ints(1,2,3,4); -- Will generate an error
ERROR 3467: Function add2or3Ints(int, int, int, int) does not exist, or
permission is denied for add2or3Ints(int, int, int, int)
HINT: No function matches the given name and argument types. You may
need to add explicit type casts
```

The error message in response to the final call to the `add2or3Ints` function was generated by Vertica, since it could not find a factory class associated with `add2or3Ints` that accepted four integer arguments. To expand `add2or3Ints` further, you could create another factory class that accepted this signature, and either change the `Add2or3ints` `ScalarFunction` class or create a totally different class to handle adding more integers together. However, adding more classes to accept each variation in the arguments quickly becomes overwhelming. In that case, you should consider creating a polymorphic UDx.

Java Example: Overloading Your UDx

The following example code demonstrates creating a User-Defined Scalar Function (UDSF) that adds two or three integers together. The `Add2or3ints` class is prepared to handle two or three arguments. It checks the number of arguments that have been passed to it, and adds all two or three of them together. The `processBlock()` method checks whether it has been called with less than 2 or more than 3 arguments. In theory, this should never happen, since Vertica only calls the UDSF if the user's function call matches a signature on one of the factory classes you create for your function. In practice, it is a good idea to perform this sanity checking, in case your (or someone else's) factory class reports that your function class accepts a set of arguments that it actually does not.

```
// You need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.multiparamexample;
// Import the entire Vertica SDK
import com.vertica.sdk.*;
// This ScalarFunction accepts two or three integer arguments. It tests
// the number of input columns to determine whether to read two or three
// arguments as input.
public class Add2or3ints extends ScalarFunction
{
    @Override
    public void processBlock(ServerInterface srvInterface,
        BlockReader argReader,
        BlockWriter resWriter)
        throws UdfException, DestroyInvocation
    {
        // See how many arguments were passed in
        int numCols = argReader.getNumCols();
```

```
// Return an error if less than two or more than 3 arguments
// were given. This error only occurs if a Factory class that
// accepts the wrong number of arguments instantiates this
// class.
if (numCols < 2 || numCols > 3) {
    throw new UdfException(0,
        "Must supply 2 or 3 integer arguments");
}

// Process all of the rows of input.
do {
    // Get the first two integer arguments from the BlockReader
    long a = argReader.getLong(0);
    long b = argReader.getLong(1);

    // Assume no third argument.
    long c = 0;

    // Get third argument value if it exists
    if (numCols == 3) {
        c = argReader.getLong(2);
    }

    // Process the arguments and come up with a result. For this
    // example, just add the three arguments together.
    long result = a+b+c;

    // Write the integer output value.
    resWriter.setLong(result);

    // Advance the output BlockWriter to the next row.
    resWriter.next();

    // Continue processing input rows until there are no more.
} while (argReader.next());
}
```

The main difference between the `Add2ints` class and the `Add2or3ints` class is the inclusion of a section that gets the number of arguments by calling `BlockReader.getNumCols()`. This class also tests the number of columns it received from Vertica to ensure it is in the range it is prepared to handle. This test will only fail if you create a `ScalarFunctionFactory` whose `getPrototype()` method defines a signature that accepts less than two or more than three arguments. This is not really necessary in this simple example, but for a more complicated class it is a good idea to test the number of columns and data types that Vertica passed your function class.

Within the `do` loop, `Add2or3ints` uses a default value of zero if Vertica sent it two input columns. Otherwise, it retrieves the third value and adds that to the other two. Your own class needs to use default values for missing input columns or alter its processing in some other way to handle the variable columns.

You must define your function class in its own source file, rather than as an inner class of one of your factory classes since Java does not allow the instantiation of an inner class from outside its containing class. Your factory class has to be available for instantiation by multiple factory classes.

Once you have created a function class or classes, you create a factory class for each signature you want your function class to handle. These factory classes can call individual function classes, or they can all call the same class that is prepared to accept multiple sets of arguments.

The following example's `createScalarFunction()` method instantiates a member of the `Add2or3ints` class.

```
// You will need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.multiparamexample;
// Import the entire Vertica SDK
import com.vertica.sdk.*;
public class Add2intsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes,
        ColumnTypes returnType)
    {
        // Accept two integers as input
        argTypes.addInt();
        argTypes.addInt();
        // writes one integer as output
        returnType.addInt();
    }
    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface)
    {
        // Instantiate the class that can handle either 2 or 3 integers.
        return new Add2or3ints();
    }
}
```

The following `ScalarFunctionFactory` subclass accepts three integers as input. It, too, instantiates a member of the `Add2or3ints` class to process the function call:

```
// You will need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.multiparamexample;
// Import the entire Vertica SDK
import com.vertica.sdk.*;
public class Add3intsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes,
        ColumnTypes returnType)
    {
        // Accepts three integers as input
        argTypes.addInt();
    }
}
```

```
    argTypes.addInt();
    argTypes.addInt();
    // Returns a single integer
    returnType.addInt();
  }
  @Override
  public ScalarFunction createScalarFunction(ServerInterface srvInterface)
  {
    // Instantiates the Add2or3ints ScalarFunction class, which is able to
    // handle either 2 or 3 integers as arguments.
    return new Add2or3ints();
  }
}
```

The factory classes and the function class or classes they call must be packaged into the same JAR file (see [Compiling and Packaging a Java Library](#) for details). If a host in the database cluster has the JDK installed on it, you could use the following commands to compile and package the example:

```
$ cd pathToJavaProject$ javac -classpath /opt/vertica/bin/VerticaSDK.jar \
> com/mycompany/multiparamexample/*.java
$ jar -cvf Add2or3intslib.jar com/vertica/sdk/BuildInfo.class \
> com/mycompany/multiparamexample/*.class
added manifest
adding: com/vertica/sdk/BuildInfo.class(in = 1202) (out= 689)(deflated 42%)
adding: com/mycompany/multiparamexample/Add2intsFactory.class(in = 677) (out= 366)(deflated 45%)
adding: com/mycompany/multiparamexample/Add2or3ints.class(in = 919) (out= 601)(deflated 34%)
adding: com/mycompany/multiparamexample/Add3intsFactory.class(in = 685) (out= 369)(deflated 46%)
```

Once you have packaged your overloaded UDX, you deploy it the same way as you do a regular UDX, except you use multiple [CREATE FUNCTION](#) statements to define the function, once for each factory class.

```
=> CREATE LIBRARY add2or3intslib as '/home/dbadmin/Add2or3intslib.jar'
-> language 'Java';
CREATE LIBRARY
=> CREATE FUNCTION add2or3ints as LANGUAGE 'Java' NAME
'com.mycompany.multiparamexample.Add2intsFactory' LIBRARY add2or3intslib;
CREATE FUNCTION
=> CREATE FUNCTION add2or3ints as LANGUAGE 'Java' NAME
'com.mycompany.multiparamexample.Add3intsFactory' LIBRARY add2or3intslib;
CREATE FUNCTION
```

You call the overloaded function the same way you call any other function.

```
=> SELECT add2or3ints(2,3);
   add2or3ints
-----
           5
(1 row)
=> SELECT add2or3ints(2,3,4);
   add2or3ints
-----
           9
```

```
(1 row)
=> SELECT add2or3ints(2,3,4,5);
ERROR 3457: Function add2or3ints(int, int, int, int) does not exist, or permission is denied for
add2or3ints(int, int, int, int)
HINT: No function matches the given name and argument types. You may need to add explicit type casts
```

The last error was generated by Vertica, not the UDX code. It returns an error if it cannot find a factory class whose signature matches the function call's signature.

Creating an overloaded UDX is useful if you want your function to accept a limited set of potential arguments. If you want to create a more flexible function, you can create a polymorphic function.

Creating a Polymorphic UDX

Polymorphic UDXs accept any number and type of argument that the user supplies. Vertica does not check the number or types of argument that the user passes to the UDX—it just passes the UDX all of the arguments supplied by the user. It is up to your polymorphic UDX's main processing function (for example, `processBlock()` in User-Defined Scalar Functions) to examine the number and types of arguments it received and determine if it can handle them.

Note: UDXs have a maximum 1600 arguments.

Polymorphic UDXs are more flexible than using multiple factory classes for your function (see [Overloading Your UDX](#)), because your function can determine at run time if it can process the arguments rather than accepting specific sets of arguments. However, your polymorphic function needs to perform more work to determine whether it can process the arguments that it has been given.

Your polymorphic UDX declares that it accepts any number of arguments in its factory's `getPrototype()` function by calling the `addAny()` function on the `ColumnTypes` object that defines its arguments. This "any parameter" argument type is the only one that your function can declare. You cannot define required arguments and then call `addAny()` to declare the rest of the signature as optional. If your function has requirements for the arguments it accepts, your process function must enforce them.

Polymorphic UDXs and Schema Search Paths

If a user does not supply a schema name as part of a UDX call, Vertica searches each schema in the schema search path for a function whose name and signature match the function call. See [Setting Search Paths](#) in the Administrator's Guide for more information about schema search paths.

Since polymorphic UDxs do not have a specific signature associated with them, Vertica initially skips them when searching for a function to handle the function call. If none of the schemas in the search path contain a UDx whose name and signature match the function call, Vertica searches the schema search path again for a polymorphic UDx whose name matches the function name in the function call.

This behavior gives precedence to UDxs whose signature exactly matches the function call. It allows you to create a "catch all" polymorphic UDx that Vertica calls only when none of the non-polymorphic UDxs with the same name have matching signatures.

This behavior may cause confusion if your users expect the first polymorphic function in the schema search path to handle a function call. To avoid confusion, you should:

- Avoid using the same name for different UDxs. You should always uniquely name UDxs unless you intend to create an overloaded UDx with multiple signatures.
- When you cannot avoid having UDxs with the same name in different schemas, always supply the schema name as part of the function call. Using the schema name prevents ambiguity and ensures that Vertica uses the correct UDx to process your function calls.

C++ Example: Polymorphic UDx

The following example shows an implementation of a `ScalarFunction` that adds together two or more integers.

```
#include "Vertica.h"
using namespace Vertica;
using namespace std;
// Adds two or more integers together.
class AddManyInts : public Vertica::ScalarFunction
{
public:
    virtual void processBlock(Vertica::ServerInterface &srvInterface,
                             Vertica::BlockReader &arg_reader,
                             Vertica::BlockWriter &res_writer)
    {
        // Always catch exceptions to prevent causing the side process or
        // Vertica itself from crashing.
        try
        {
            // Find the number of arguments sent.
            size_t numCols = arg_reader.getNumCols();

            // Make sure at least 2 arguments were supplied
            if (numCols < 2)
                vt_report_error(0, "Function expects at least 2 integer parameters");

            // Make sure all types are ints
            const SizedColumnTypes &inTypes = arg_reader.getTypeMetaData();
            for (int param=0; param < (int)numCols; param++) {
```

```
const VerticaType &t = inTypes.getColumnType(param);
if (!t.isInt())
{
    string typeDesc = t.getPrettyPrintStr();
    // Report that the user supplied a non-integer value.
    vt_report_error(0, "Function expects all arguments to be "
        "INTEGER. Argument %d was %s", param+1,
        typeDesc.c_str());
}
}
do
{ // total up the arguments and write out the total.
    vint total = 0;
    int x;
    // Loop over all params, adding them up.
    for (x=0; x<(int)numCols; x++) {
        total += arg_reader.getIntRef(x);
    }
    res_writer.setInt(total);
    res_writer.next();
} while (arg_reader.next());
} catch(exception& e) {
    // Standard exception. Quit.
    vt_report_error(0, "Exception while processing partition: [%s]",
        e.what());
}
}
};
// Defines the AddMany function.
class AddManyIntsFactory : public Vertica::ScalarFunctionFactory
{
    // Return the function object to process the data.
    virtual Vertica::ScalarFunction *createScalarFunction(
        Vertica::ServerInterface &srvInterface)
    { return vt_createFuncObj(srvInterface.allocator, AddManyInts); }
    // Define the number and types of arguments that this function accepts
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
        Vertica::ColumnTypes &argTypes,
        Vertica::ColumnTypes &returnType)
    {
        argTypes.addAny(); // Must be only argument type.
        returnType.addInt();
    }
};
RegisterFactory(AddManyIntsFactory);
```

Most of the work in the example is done by the `ScalarFunction.processBlock()` function. It performs two checks on the arguments that have been passed in through the `BlockReader` object:

- Ensures there are at least two arguments
- Checks the data type of all arguments to ensure they are all integers

Once the checks are performed, the example processes the block of data by looping over the arguments and adding them together.

You assign a SQL name to your polymorphic UDX using the same statement you use to assign one to a non-polymorphic UDX. The following demonstration shows how you load and call the polymorphic function from the example.

```
=> CREATE LIBRARY addManyIntsLib AS '/home/dbadmin/AddManyInts.so';
CREATE LIBRARY
=> CREATE FUNCTION addManyInts AS NAME 'AddManyIntsFactory' LIBRARY addManyIntsLib FENCED;
CREATE FUNCTION
=> SELECT addManyInts(1,2);
  addManyInts
-----
          3
(1 row)
=> SELECT addManyInts(1,2,3,40,50,60,70,80,900);
  addManyInts
-----
        1206
(1 row)
=> SELECT addManyInts(1); -- Too few parameters
ERROR 3412: Failure in UDX RPC call InvokeProcessBlock(): Error calling
processBlock() in User-Defined Object [addManyInts] at
[AddManyInts.cpp:51], error code: 0, message: Exception while processing
partition: [Function expects at least 2 integer parameters]
=> SELECT addManyInts(1,2.232343); -- Wrong data type
ERROR 3412: Failure in UDX RPC call InvokeProcessBlock(): Error
calling processBlock() in User-Defined Object [addManyInts] at
[AddManyInts.cpp:51], error code: 0, message: Exception while
processing partition: [Function expects all arguments to be INTEGER.
Argument 2 was Numeric(7,6)]
```

Notice that the errors returned by last two calls to the function were generated by the `processBlock()` function. It is up to your UDX to ensure that the user supplies the correct number and types of arguments to your function and exit with an error if it cannot process them.

Java Example: Polymorphic UDX

The following example shows an implementation of a `ScalarFunctionFactory` class with an inner `ScalarFunction` class that adds together two or more integers.

```
// You will need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.multiparamexample;
// Import the entire Vertica SDK
import com.vertica.sdk.*;
// Factory class to create polymorphic UDSF that adds all of the integer
// arguments it receives and returns a sum.
public class AddManyIntsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes,
        ColumnTypes returnType)
```

```
{
// Accepts any number and type or arguments. The ScalarFunction
// class handles parsing the arguments.
argTypes.addAny();
// writes one integer as output
returnType.addInt();
}
// This polymorphic ScalarFunction adds all of the integer arguments passed
// to it. Returns an error if there are less than two arguments, or if one
// argument is not an integer.
public class AddManyInts extends ScalarFunction
{
@Override
public void processBlock(ServerInterface srvInterface,
                        BlockReader argReader,
                        BlockWriter resWriter)
    throws UdfException, DestroyInvocation
{
// See how many arguments were passed in
int numCols = argReader.getNumCols();

// Return an error if less than two arguments were given.
if (numCols < 2) {
    throw new UdfException(0,
        "Must supply at least 2 integer arguments");
}

// Make sure all input columns are integer.
SizedColumnTypes inTypes = argReader.getTypeMetaData();
for (int param = 0; param < numCols; param++) {
    VerticaType paramType = inTypes.getColumnType(param);
    if (!paramType.isInt()) {
        throw new UdfException(0, "Error: Argument " + (param+1) +
            " was not an integer. All arguments must be integer.");
    }
}

// Process all of the rows of input.
do {
    long total = 0; // Hold the running total of arguments

    // Get all of the arguments and add them up
    for (int x = 0; x < numCols; x++) {
        total += argReader.getLong(x);
    }

    // Write the integer output value.
    resWriter.setLong(total);

    // Advance the output BlockWriter to the next row.
    resWriter.next();

    // Continue processing input rows until there are no more.
} while (argReader.next());
}
}

@Override
public ScalarFunction createScalarFunction(ServerInterface srvInterface)
{
```

```
// Instantiate the polymorphic UDF class.  
return new AddManyInts();  
}  
}
```

The `ScalarFunctionFactory.getPrototype()` method calls the `addAny()` method to declare that the UDSF is polymorphic.

Most of the work in the example is done by the `ScalarFunction.processBlock()` method. It performs two checks on the arguments that have been passed in through the `BlockReader` object:

- There are at least two arguments.
- The data type of all arguments are integers.

It is up to your polymorphic UDX to determine that all of the input passed to it is valid.

Once the `processBlock()` method validates its arguments, it loops over them, adding them together.

You assign a SQL name to your polymorphic UDX using the same statement you use to assign one to a non-polymorphic UDX. The following demonstration shows how you load and call the polymorphic function from the example.

```
=> CREATE LIBRARY addmanyintslib AS '/home/dbadmin/AddManyIntsLib.jar'  
-> LANGUAGE 'Java';  
CREATE LIBRARY  
=> CREATE FUNCTION addmanyints AS LANGUAGE 'Java' NAME  
-> 'com.mycompany.multiparamexample.AddManyIntsFactory' LIBRARY addmanyintslib;  
CREATE FUNCTION  
=> SELECT addmanyints(1,2,3,4,5,6,7,8,9,10);  
  addmanyints  
-----  
          55  
(1 row)  
=> SELECT addmanyints(1); --Too few parameters  
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error in User  
Defined Object [addmanyints], error code: 0  
com.vertica.sdk.UdfException: Must supply at least 2 integer arguments  
  at  
com.mycompany.multiparamexample.AddManyIntsFactory$AddManyInts.processBlock  
(AddManyIntsFactory.java:39)  
  at com.vertica.udxfence.UDXExecContext.processBlock(UDXExecContext.java:700)  
  at com.vertica.udxfence.UDXExecContext.run(UDXExecContext.java:173)  
  at java.lang.Thread.run(Thread.java:662)  
=> SELECT addmanyints(1,2,3,14159); --Non-integer parameter  
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error in User  
Defined Object [addmanyints], error code: 0  
com.vertica.sdk.UdfException: Error: Argument 3 was not an integer. All  
arguments must be integer.  
  at  
com.mycompany.multiparamexample.AddManyIntsFactory$AddManyInts.processBlock  
(AddManyIntsFactory.java:48)
```

```
at com.vertica.udxfence.UDxExecContext.processBlock(UDxExecContext.java:700)
at com.vertica.udxfence.UDxExecContext.run(UDxExecContext.java:173)
at java.lang.Thread.run(Thread.java:662)
```

R Example: Polymorphic UDx

The following example shows an implementation of a Transform Function (UDTF) that performs kmeans clustering on one or more input columns.

```
kmeansPoly <- function(v.data.frame,v.param.list) {
  # Computes clusters using the kmeans algorithm.
  #
  # Input: A dataframe and a list of parameters.
  # Output: A dataframe with one column that tells the cluster to which each data
  #        point belongs.
  # Args:
  # v.data.frame: The data from Vertica cast as an R data frame.
  # v.param.list: List of function parameters.
  #
  # Returns:
  # The cluster associated with each data point.
  # Ensure k is not null.
  if(!is.null(v.param.list[["k"]])) {
    number_of_clusters <- as.numeric(v.param.list[["k"]])
  } else {
    stop("k cannot be NULL! Please use a valid value.")
  }
  # Run the kmeans algorithm.
  kmeans_clusters <- kmeans(v.data.frame, number_of_clusters)
  final.output <- data.frame(kmeans_clusters$cluster)
  return(final.output)
}

kmeansFactoryPoly <- function() {
  # This function tells Vertica the name of the R function,
  # and the polymorphic parameters.
  list(name=kmeansPoly, udxttype=c("transform"), intype=c("any"),
        outtype=c("int"), parameterstypecallback=kmeansParameters)
}

kmeansParameters <- function() {
  # Callback function for the parameter types.
  function.parameters <- data.frame(datatype=rep(NA, 1), length=rep(NA,1),
                                    scale=rep(NA,1), name=rep(NA,1))
  function.parameters[1,1] = "int"
  function.parameters[1,4] = "k"
  return(function.parameters)
}
```

The polymorphic R function declares it accepts any number of arguments in its factory function by specifying "any" as the argument to the `intype` parameter and optionally the `outtype` parameter. If you define "any" argument for `intype` or `outtype`, then it is the only type that your function can declare for the respective parameter. You cannot define required arguments

and then call "any" to declare the rest of the signature as optional. If your function has requirements for the arguments it accepts, your process function must enforce them.

The `outtypecallback` method is used to indicate the argument types and sizes it has been called with, and is expected to indicate the types and sizes that the function returns. The `outtypecallback` method can also be used to check for unsupported types and/or number of arguments. For example, the function may require only integers, with no more than 10 of them.

You assign a SQL name to your polymorphic UDX using the same statement you use to assign one to a non-polymorphic UDX. The following statements show how you load and call the polymorphic function from the example.

```
=> CREATE LIBRARY rlib2 AS '/home/dbadmin/R_UDx/poly_kmeans.R' LANGUAGE 'R';
CREATE LIBRARY
=> CREATE TRANSFORM FUNCTION kmeansPoly AS LANGUAGE 'R' name 'kmeansFactoryPoly' LIBRARY rlib2;
CREATE FUNCTION
=> SELECT spec, kmeansPoly(s1,sw,p1,pw USING PARAMETERS k = 3)
      OVER(PARTITION BY spec) AS Clusters
      FROM iris;
      spec      | Clusters
-----+-----
Iris-setosa    |         1
Iris-setosa    |         1
Iris-setosa    |         1
Iris-setosa    |         1
.
.
.
(150 rows)
```

UDx Parameters

Parameters let you define arguments for your UDxs that remain constant across all of the rows processed by the SQL statement that calls your UDx. Typically, your UDxs accept arguments that come from columns in a SQL statement. For example, in the following SQL statement, the arguments `a` and `b` to the `add2ints` UDSF change value for each row processed by the `SELECT` statement:

```
=> SELECT a, b, add2ints(a,b) AS 'sum' FROM example;
a | b | sum
---+---+---
1 | 2 | 3
3 | 4 | 7
5 | 6 | 11
7 | 8 | 15
9 | 10 | 19
(5 rows)
```

Parameters remain constant for all the rows your UDx processes. You can also make parameters optional so that if the user does not supply it, your UDx uses a default value. For example, the following example demonstrates calling a UDSF named `add2intsWithConstant` that has a single parameter value named `constant` whose value is added to each the arguments supplied in each row of input:

```
=> SELECT a, b, add2intsWithConstant(a, b USING PARAMETERS constant=42)
      AS 'a+b+42' from example;
a | b | a+b+42
---+---+---
1 | 2 | 45
3 | 4 | 49
5 | 6 | 53
7 | 8 | 57
9 | 10 | 61
(5 rows)
```

Note: When calling a UDx with parameters, there is no comma between the last argument and the `USING PARAMETERS` clause.

The topics in this section explain how to develop UDxs that accept parameters.

Defining the Parameters Your UDx Accepts

You define the parameters that your UDx accepts in its factory class (`ScalarFunctionFactory`, `AggregateFunctionFactory`, and so on) by implementing

`getParameterType()`. This method is similar to `getReturnType()`: you call data-type-specific methods on a `SizedColumnTypes` object that is passed in as a parameter. Each function call sets the name, data type, and width or precision (if the data type requires it) of the parameter.

Setting Parameter Properties (C++ Only)

When you add parameters to the `getParameterType()` function using the C++ API, you can also set properties for each parameter. For example, you can define a parameter as being required by the UDX. Doing so lets the Vertica server know that every UDX invocation must provide the specified parameter, or the query fails.

By passing an object to the `SizedColumnTypes::Properties` class, you can define the following four parameter properties:

Parameter	Type	Description
<code>visible</code>	BOOLEAN	If set to TRUE, the parameter appears in the USER_FUNCTION_PARAMETERS table. You may want to set this to FALSE to declare a parameter for internal use only.
<code>required</code>	BOOLEAN	If set to TRUE: <ul style="list-style-type: none">• The parameter is required when invoking the UDX.• Invoking the UDX without supplying the parameter results in an error, and the UDX does not run.
<code>canBeNull</code>	BOOLEAN	If set to TRUE, the parameter can have a NULL value. If set to FALSE, make sure that the supplied parameter does not contain a NULL value when invoking the UDX. Otherwise, an error results, and the UDX does not run.
<code>comment</code>	VARCHAR (128)	A comment to describe the parameter. If you exceed the 128 character limit, Vertica generates an error when you run the <code>CREATE_FUNCTION</code> command. Additionally, if you replace the existing function definition in the <code>comment</code> parameter, make sure that the new definition does not exceed 128 characters. Otherwise, you delete all existing entries in the <code>USER_FUNCTION_PARAMETERS</code> table related to the UDX.

Setting Parameter Properties (R Only)

When using parameters in your R UDX, you must specify a field in the factory function called `parametertypecallback`. This field points to the callback function that defines the parameters expected by the function. The callback function defines a four-column data frame with the following properties:

Parameter	Type	Description
<code>datatype</code>	<code>VARCHAR(128)</code>	The data type of the parameter.
<code>length</code>	<code>INT</code>	The dimension of the parameter.
<code>scale</code>	<code>INT</code>	The proportional dimensions of the parameter.
<code>name</code>	<code>VARCHAR(128)</code>	The name of the parameter.

If any of the columns are left blank (or the `parametertypecallback` function is omitted), then Vertica uses default values.

For more information, see [parametertypecallback Function](#).

Getting Parameter Values in UDXs

Your UDX uses the parameter values it declared in its factory class (see [Defining the Parameters Your UDX Accepts](#)) in its function class's processing method (for example, `processBlock()` or `processPartition()`). It gets its parameter values from a `ParamReader` object, which is available from the `ServerInterface` object that is passed to your processing method.

Reading parameters from this object is similar to reading argument values from `BlockReader` or `PartitionReader` objects: you call a data-type-specific function with the name of the parameter to retrieve its value. For example, in C++:

```
// Get the parameter reader from the ServerInterface to see if there are supplied parameters.
ParamReader paramReader = srvInterface.getParamReader();
// Get the value of an int parameter named constant.
const vint constant = paramReader.getIntRef("constant");
```

Note: String data values do not have any of their escape characters processed before they are passed to your function. Therefore, your function may need to process the escape sequences itself if it needs to operate on unescaped character values.

Using Parameters in the Factory Class

In addition to using parameters in your UDX function class, you can also access the parameters in the factory class. You may want to access the parameters to let the user control the input or output values of your function in some way. For example, your UDX can have a parameter that lets the user choose to have your UDX return a single- or double-precision value. The process of accessing parameters in the factory class is the same as accessing it in the function class: get a `ParamReader` object from the `ServerInterface`'s `getParamReader()` method, then read the parameter values.

Testing Whether the User Supplied Parameter Values

Unlike its handling of arguments, Vertica does not immediately return an error if a user's function call does not include a value for a parameter defined by your UDX's factory class. This means that your function can attempt to read a parameter value that the user did not supply. If it does so, by default Vertica returns a non-existent parameter warning to the user, and the query containing the function call continues.

If you want your parameter to be optional, you can test whether the user supplied a value for the parameter before attempting to access its value. Your function determines if a value exists for a particular parameter by calling the `ParamReader`'s `containsParameter()` method with the parameter's name. If this call returns `true`, your function can safely retrieve the value. If this call returns `false`, your UDX can use a default value or change its processing in some other way to compensate for not having the parameter value. As long as your UDX does not try to access the non-existent parameter value, Vertica does not generate an error or warning about missing parameters.

Note: If the user passes your UDX a parameter that it has not defined, by default Vertica issues a warning that the parameter is not used. It still executes the SQL statement, ignoring the parameter. You can change this behavior by altering the `StrictUDXParameterChecking` configuration parameter.

See [C++ Example: Defining Parameters](#) for an example.

Calling UDXs with Parameters

You pass parameters to a UDX by adding a USING PARAMETERS clause in the function call after the last argument.

- Do *not* insert a comma between the last argument and the USING PARAMETERS clause.
- After the USING PARAMETERS clause, add one or more parameter definitions, in the following form:

```
<parameter name> = <parameter value>
```

- Separate parameter definitions by commas.

Parameter values can be a constant expression (for example `1234 + SQRT(5678)`). You cannot use volatile functions (such as [RANDOM](#)) in the expression, because they do not return a constant value. If you do supply a volatile expression as a parameter value, by default, Vertica returns an incorrect parameter type warning. Vertica then tries to run the UDX without the parameter value. If the UDX requires the parameter, it returns its own error, which cancels the query.

Calling a UDX with a Single Parameter

The following example demonstrates how you can call the `Add2intsWithConstant` UDSF example shown in [C++ Example: Defining Parameters](#):

```
=> SELECT a, b, Add2intsWithConstant(a, b USING PARAMETERS constant=42) AS 'a+b+42' from example;
 a | b | a+b+42
---+---+-----
 1 | 2 |      45
 3 | 4 |      49
 5 | 6 |      53
 7 | 8 |      57
 9 | 10 |     61
(5 rows)
```

To remove the first instance of the number 3, you can call the `RemoveSymbol` UDSF example:

```
=> SELECT '3re3mo3ve3sy3mb3o1' original_string, RemoveSymbol('3re3mo3ve3sy3mb3o1' USING PARAMETERS
symbol='3');
 original_string | RemoveSymbol
-----+-----
 3re3mo3ve3sy3mb3o1 | re3mo3ve3sy3mb3o1
(1 row)
```

Calling a UDX with Multiple Parameters

The following example shows how you can call a version of the tokenize UDTF. This UDTF includes parameters to limit the shortest allowed word and force the words to be output in uppercase. Separate multiple parameters with commas.

```
=> SELECT url, tokenize(description USING PARAMETERS minLength=4, uppercase=true) OVER (partition by
url) FROM T;
      url      | words
-----+-----
www.amazon.com | ONLINE
www.amazon.com | RETAIL
www.amazon.com | MERCHANT
www.amazon.com | PROVIDER
www.amazon.com | CLOUD
www.amazon.com | SERVICES
www.hpe.com    | LEADING
www.hpe.com    | PROVIDER
www.hpe.com    | COMPUTER
www.hpe.com    | HARDWARE
www.hpe.com    | IMAGING
www.hpe.com    | SOLUTIONS
www.vertica.com | WORLD'S
www.vertica.com | FASTEST
www.vertica.com | ANALYTIC
www.vertica.com | DATABASE
(16 rows)
```

The following example calls the RemoveSymbol UDSF. By changing the value of the optional parameter, *n*, you can remove all instances of the number 3:

```
=> SELECT '3re3mo3ve3sy3mb3o1' original_string, RemoveSymbol('3re3mo3ve3sy3mb3o1' USING PARAMETERS
symbol='3', n=6);
original_string | RemoveSymbol
-----+-----
3re3mo3ve3sy3mb3o1 | removesymbol
(1 row)
```

Calling a UDX with Optional or Incorrect Parameters

You can optionally add the Add2intsWithConstant UDSF's constant parameter. Calling this constraint without the parameter does not return an error or warning:

```
=> SELECT a,b,Add2intsWithConstant(a, b) AS 'sum' FROM example;
 a | b | sum
---+---+---
 1 | 2 |  3
 3 | 4 |  7
 5 | 6 | 11
```

```
7 | 8 | 15
9 | 10 | 19
(5 rows)
```

Although calling a UDX with incorrect parameters generates a warning, by default, the query still runs. For further information on setting the behavior of your UDX when you supply incorrect parameters, see [Specifying the Behavior of Passing Unregistered Parameters](#).

```
=> SELECT a, b, add2intsWithConstant(a, b USING PARAMETERS wrongparam=42) AS 'result' from example;
WARNING 4332: Parameter wrongparam was not registered by the function and cannot
be coerced to a definite data type
 a | b | result
---+---+-----
 1 | 2 |      3
 3 | 4 |      7
 5 | 6 |     11
 7 | 8 |     15
 9 | 10 |     19
(5 rows)
```

Specifying the Behavior of Passing Unregistered Parameters

By default, Vertica issues a warning message when you pass a UDX an unregistered parameter. An *unregistered parameter* is one that you did not declare in the `getParameterType()` method.

You can control the behavior of your UDX when you pass it an unregistered parameter by altering the `StrictUDXParameterChecking` configuration parameter.

Unregistered Parameter Behavior Settings

You can specify the behavior of your UDX in response to one or more unregistered parameters. To do so, set the `StrictUDXParameterChecking` configuration parameter to one of the following values:

Value	Description
0	Allows unregistered parameters to be accessible to the UDX. The <code>ParamReader</code> class's <code>getType()</code> method determines the data type of the unregistered parameter. Vertica does not display any warning or error message.

Value	Description
1	[Default Value] Ignores the unregistered parameter and allows the function to run. Vertica displays a warning message.
2	Returns an error and does not allow the function to run.

Examples

The following examples demonstrate the behavior you can specify using different values with the `StrictUDxParameterChecking` parameter.

View the Current Value of `StrictUDxParameterChecking`

To view the current value of the `StrictUDxParameterChecking` configuration parameter, run the following query:

```
=> \x
Expanded display is on.
=> SELECT * FROM configuration_parameters WHERE parameter_name = 'StrictUDxParameterChecking';
-[ RECORD 1 ]-----+-----
node_name          | ALL
parameter_name     | StrictUDxParameterChecking
current_value      | 1
restart_value      | 1
database_value     | 1
default_value      | 1
current_level      | DATABASE
restart_level      | DATABASE
is_mismatch        | f
groups             |
allowed_levels     | DATABASE
superuser_only     | f
change_under_support_guidance | f
change_requires_restart | f
description        | Sets the behavior to deal with undeclared Udx function parameters
```

Change the Value of `StrictUDxParameterChecking`

You can change the value of the `StrictUDxParameterChecking` configuration parameter at the database, node, or session level. For example, you can change the value to '0' to specify that unregistered parameters can pass to the Udx without displaying a warning or error message:

```
=> ALTER DATABASE mydb SET StrictUDxParameterChecking = 0;
ALTER DATABASE
```

Invalid Parameter Behavior with RemoveSymbol

The following example demonstrates how to call the RemoveSymbol UDSF example. The RemoveSymbol UDSF has a required parameter, `symbol`, and an optional parameter, `n`. In this case, you do not use the optional parameter.

If you pass both `symbol` and an additional parameter called `wrongParam`, which is not declared in the UDX, the behavior of the UDX changes corresponding to the value of `StrictUDXParameterChecking`.

When you set `StrictUDXParameterChecking` to '0', the UDX runs normally without a warning. Additionally, `wrongParam` becomes accessible to the UDX through the `ParamReader` object of the `ServerInterface` object:

```
=> ALTER DATABASE mydb SET StrictUDXParameterChecking = 0;
ALTER DATABASE

=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS
symbol='3', wrongParam='x');
  original_string | RemoveSymbol
-----+-----
3re3mo3ve3sy3mb3ol | re3mo3ve3sy3mb3ol
(1 row)
```

When you set `StrictUDXParameterChecking` to '1', the UDX ignores `wrongParam` and runs normally. However, it also issues a warning message:

```
=> ALTER DATABASE mydb SET StrictUDXParameterChecking = 1;
ALTER DATABASE

=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS
symbol='3', wrongParam='x');
WARNING 4320: Parameter wrongParam was not registered by the function and cannot be coerced to a
definite data type
  original_string | RemoveSymbol
-----+-----
3re3mo3ve3sy3mb3ol | re3mo3ve3sy3mb3ol
(1 row)
```

When you set `StrictUDXParameterChecking` to '2', the UDX encounters an error when it tries to call `wrongParam` and does not run. Instead, it generates an error message:

```
=> ALTER DATABASE mydb SET StrictUDXParameterChecking = 2;
ALTER DATABASE

=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS
symbol='3', wrongParam='x');
ERROR 0: Parameter wrongParam was not registered by the function
```

See Also

- [Setting Configuration Parameter Values](#)

User-Defined Session Parameters

User-defined session parameters allow you to write more generalized parameters than what Vertica provides. You can configure user-defined session parameters in these ways:

- From the client (for example, through the ALTER SESSION command)
- Through the UDX itself

A user-defined session parameter can be passed into any type of UDX supported by Vertica. You can also set parameters for your UDX at the session level. By specifying a user-defined session parameter, you can have the state of a parameter saved continuously. Vertica saves the state of the parameter even when the UDX is invoked multiple times during a single session.

The RowCount example uses a user-defined session parameter. This parameter counts the total number of rows processed by the UDX each time it runs. RowCount then displays the aggregate number of rows processed for all executions. See [C++ Example: Using Session Parameters](#) and [Java Example: Using Session Parameters](#) for implementations.

Viewing the User-Defined Session Parameter

Enter the following command to see the value of the session parameter:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
(0 rows)
```

Now, execute the UDX:

```
=> SELECT RowCount(5,5);
RowCount
-----
10
(1 row)
```

Again, enter the command to see the value of the session parameter:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
public | UDSession | rowcount | 1
(1 row)
```

Because the UDX has processed one row, the value of RowCount is now 1. Running the UDX two more times should increase the value of RowCount by 2:

```
=> SELECT RowCount(10,10);
RowCount
-----
20
(1 row)
=> SELECT RowCount(15,15);
RowCount
-----
30
(1 row)
```

You have now executed the UDX three times, obtaining the sum of 5 + 5, 10 + 10, and 15 + 15. Now, check the value of RowCount.

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
public | UDSession | rowcount | 3
(1 row)
```

Altering the User-Defined Session Parameter

You can also manually alter the value of RowCount. To do so, enter the following command:

```
=> ALTER SESSION SET UDPARAMETER FOR UDSession RowCount = 25;
ALTER SESSION
```

Check the value of RowCount:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
public | UDSession | rowcount | 25
(1 row)
```

Clearing the User-Defined Session Parameter

From the client:

To clear the current value of RowCount, enter the following command:

```
=> ALTER SESSION CLEAR UDPARAMETER FOR UDSession RowCount;
ALTER SESSION
```

Verify that RowCount has been cleared:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
(0 rows)
```

Through the UDx in C++:

You can set the session parameter to clear through the UDx itself. For example, to clear RowCount when its value reaches 10 or greater do the following:

1. Remove the following line from the `destroy()` method in RowCount:

```
udParams.getUDSessionParamWriter("library").getStringRef("rowCount").copy(i_as_string);
```

2. Replace the removed line from the `destroy()` method with the following code:

```
if (rowCount < 10)
{
udParams.getUDSessionParamWriter("library").getStringRef("rowCount").copy(i_as_string);
}
else
{
udParams.getUDSessionParamWriter("library").clearParameter("rowCount");
}
```

3. To see the UDx clear the session parameter, set RowCount to a value of 9:

```
=> ALTER SESSION SET UDPARAMETER FOR UDSession RowCount = 9;
ALTER SESSION
```

4. Check the value of RowCount:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
public | UDSession | rowcount | 9
(1 row)
```

5. Invoke RowCount so that its value becomes 10:

```
=> SELECT RowCount(15,15);
RowCount
-----
30
```

```
(1 row)
```

6. Check the value of RowCount. Because the value has reached 10, the threshold specified in the UDx, expect that RowCount is cleared:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
(0 rows)
```

As expected, RowCount is cleared.

Through the UDx in Java:

1. Remove the following lines from the `destroy()` method in RowCount:

```
udParams.getUDSessionParamWriter("library").setString("rowCount", Integer.toString(rowCount));
srvInterface.log("RowNumber processed %d records", count);
```

2. Replace the removed lines from the `destroy()` method with the following code:

```
if (rowCount < 10)
{
udParams.getUDSessionParamWriter("library").setString("rowCount", Integer.toString(rowCount));
srvInterface.log("RowNumber processed %d records", count);
}
else
{
udParams.getUDSessionParamWriter("library").clearParameter("rowCount");
}
```

3. To see the UDx clear the session parameter, set RowCount to a value of 9:

```
=> ALTER SESSION SET UDPARAMETER FOR UDSession RowCount = 9;
ALTER SESSION
```

4. Check the value of RowCount:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
public | UDSession | rowcount | 9
(1 row)
```

5. Invoke RowCount so that its value becomes 10:

```
=> SELECT RowCount(15,15);
RowCount
-----
          30
(1 row)
```

6. Check the value of RowCount. Since the value has reached 10, the threshold specified in the UDx, expect that RowCount is cleared:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
-----+-----+-----+-----
(0 rows)
```

As expected, RowCount is cleared.

Read-Only and Hidden Session Parameters

If you don't want a parameter to be set anywhere except in the UDx, you can make it read-only. If, additionally, you don't want a parameter to be visible in the client, you can make it hidden.

To make a parameter read-only, meaning that it cannot be set in the client, but can be viewed, add a single underscore before the parameter's name. For example, to make rowCount read-only, change all instances in the UDx of "rowCount" to "_rowCount".

To make a parameter hidden, meaning that it cannot be viewed in the client nor set, add two underscores before the parameter's name. For example, to make rowCount hidden, change all instances in the UDx of "rowCount" to "__rowCount".

See Also

[User-Defined Session Parameters](#)

C++ Example: Defining Parameters

The following code fragment demonstrates adding a single parameter to the C++ add2ints UDSF example. The `getParameterType()` function defines a single integer parameter that is named constant.

```
class Add2intsWithConstantFactory : public ScalarFunctionFactory
{
```

```
// Return an instance of Add2ints to perform the actual addition.
virtual ScalarFunction *createScalarFunction(ServerInterface &interface)
{
    // Calls the vt_createFuncObj to create the new Add2ints class instance.
    return vt_createFuncObj(interface.allocator, Add2intsWithConstant);
}
// Report the argument and return types to Vertica.
virtual void getPrototype(ServerInterface &interface,
    ColumnTypes &argTypes,
    ColumnTypes &returnType)
{
    // Takes two ints as inputs, so add ints to the argTypes object.
    argTypes.addInt();
    argTypes.addInt();
    // Returns a single int.
    returnType.addInt();
}
// Defines the parameters for this UDSF. Works similarly to defining arguments and return types.
virtual void getParameterType(ServerInterface &srvInterface,
    SizedColumnTypes &parameterTypes)
{
    // One int parameter named constant.
    parameterTypes.addInt("constant");
}
};
RegisterFactory(Add2intsWithConstantFactory);
```

See the Vertica SDK entry for `SizedColumnTypes` for a full list of the data-type-specific functions you can call to define parameters.

The following code fragment demonstrates using the parameter value. The `Add2intsWithConstant` class defines a function that adds two integer values. If the user supplies it, the function also adds the value of the optional integer parameter named `constant`.

```
/**
 * A UDSF that adds two numbers together with a constant value.
 */
class Add2intsWithConstant : public ScalarFunction
{
public:
    // Processes a block of data sent by Vertica.
    virtual void processBlock(ServerInterface &srvInterface,
        BlockReader &arg_reader,
        BlockWriter &res_writer)
    {
        try
        {
            // The default value for the constant parameter is 0.
            vint constant = 0;

            // Get the parameter reader from the ServerInterface to see if there are supplied parameters.
            ParamReader paramReader = srvInterface.getParamReader();
            // See if the user supplied the constant parameter.
            if (paramReader.containsParameter("constant"))
                // There is a parameter, so get its value.
                constant = paramReader.getIntRef("constant");
        }
    }
};
```

```
// While we have input to process:
do
{
    // Read the two integer input parameters by calling the BlockReader.getIntRef class function.
    const vint a = arg_reader.getIntRef(0);
    const vint b = arg_reader.getIntRef(1);
    // Add arguments plus constant.
    res_writer.setInt(a+b+constant);
    // Finish writing the row, and advance to the next output row.
    res_writer.next();
    // Continue looping until there are no more input rows.
}
while (arg_reader.next());
}
catch (exception& e)
{
    // Standard exception. Quit.
    vt_report_error(0, "Exception while processing partition: %s",
        e.what());
}
}
};
```

C++ Example: Using Session Parameters

The RowCount example uses a user-defined session parameter, also called RowCount. This parameter counts the total number of rows processed by the UDX each time it runs. RowCount then displays the aggregate number of rows processed for all executions.

```
#include <string>
#include <sstream>
#include <iostream>
#include "Vertica.h"
#include "VerticaUDx.h"

using namespace Vertica;

class RowCount : public Vertica::ScalarFunction
{
private:
    int rowCount;
    int count;

public:

    virtual void setup(Vertica::ServerInterface &srvInterface, const Vertica::SizedColumnTypes &argTypes) {
        ParamReader pSessionParams = srvInterface.getUDSessionParamReader("library");
        std::string rCount = pSessionParams.containsParameter("rowCount")?
            pSessionParams.getStringRef("rowCount").str(): "0";
        rowCount=atoi(rCount.c_str());
    }

    virtual void processBlock(Vertica::ServerInterface &srvInterface, Vertica::BlockReader &arg_reader, Vertica::BlockWriter
    &res_writer) {
```

```
count = 0;
if(arg_reader.getNumCols() != 2)
    vt_report_error(0, "Function only accepts two arguments, but %zu provided", arg_reader.getNumCols());

do {
    const Vertica::vint a = arg_reader.getIntRef(0);
    const Vertica::vint b = arg_reader.getIntRef(1);
    res_writer.setInt(a+b);
    count++;
    res_writer.next();
} while (arg_reader.next());

srvInterface.log("count %d", count);

}

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SessionParamWriterMap
&udParams) {
    rowCount = rowCount + count;

    std::ostringstream s;
    s << rowCount;
    const std::string i_as_string(s.str());

    udParams.getUDSessionParamWriter("library").getStringRef("rowCount").copy(i_as_string);

}
};

class RowCountsInfo : public Vertica::ScalarFunctionFactory {
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
    { return Vertica::vt_createFuncObject<RowCount>(srvInterface.allocator);
    }

    virtual void getPrototype(Vertica::ServerInterface &srvInterface, Vertica::ColumnTypes &argTypes, Vertica::ColumnTypes
&returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }
};

RegisterFactory(RowCountsInfo);
```

Java Example: Defining Parameters

The following code fragment demonstrates adding a single parameter to the Java add2ints UDSF example. The `getParameterType()` method defines a single integer parameter that is named constant.

```
package com.mycompany.example;
import com.vertica.sdk.*;
public class Add2intsWithConstantFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes,
        ColumnTypes returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    @Override
    public void getReturnType(ServerInterface srvInterface,
        SizedColumnTypes argTypes,
        SizedColumnTypes returnType)
    {
        returnType.addInt("sum");
    }

    // Defines the parameters for this UDSF. Works similarly to defining
    // arguments and return types.
    public void getParameterType(ServerInterface srvInterface,
        SizedColumnTypes parameterTypes)
    {
        // One INTEGER parameter named constant
        parameterTypes.addInt("constant");
    }

    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface)
    {
        return new Add2intsWithConstant();
    }
}
```

See the Vertica Java SDK entry for `SizedColumnTypes` for a full list of the data-type-specific methods you can call to define parameters.

Java Example: Using Session Parameters

The `RowCount` example uses a user-defined session parameter, also called `RowCount`. This parameter counts the total number of rows processed by the UDX each time it runs. `RowCount` then displays the aggregate number of rows processed for all executions.

```
package com.mycompany.example;

import com.vertica.sdk.*;

public class RowCountFactory extends ScalarFunctionFactory {
```

```
@Override
public void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType)
{
    argTypes.addInt();
    argTypes.addInt();
    returnType.addInt();
}

public class RowCount extends ScalarFunction {

    private Integer count;
    private Integer rowCount;

    // In the setup method, you look for the rowCount parameter. If it doesn't exist, it is created.
    // Look in the default namespace which is "library," but it could be anything else, most likely "public" if not "library".
    @Override
    public void setup(ServerInterface srvInterface, SizedColumnTypes argTypes) {
        count = new Integer(0);
        ParamReader pSessionParams = srvInterface.getUDSessionParamReader("library");
        String rCount = pSessionParams.containsParameter("rowCount")?
        pSessionParams.getString("rowCount"): "0";
        rowCount = Integer.parseInt(rCount);
    }

    @Override
    public void processBlock(ServerInterface srvInterface, BlockReader arg_reader, BlockWriter res_writer)
        throws UdfException, DestroyInvocation {
        do {
            ++count;
            long a = arg_reader.getLong(0);
            long b = arg_reader.getLong(1);

            res_writer.setLong(a+b);
            res_writer.next();
        } while (arg_reader.next());
    }

    @Override
    public void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes, SessionParamWriterMap udParams){
        rowCount = rowCount+count;
        udParams.getUDSessionParamWriter("library").setString("rowCount", Integer.toString(rowCount));
        srvInterface.log("RowNumber processed %d records", count);
    }
}

@Override
public ScalarFunction createScalarFunction(ServerInterface srvInterface){
    return new RowCount();
}
}
```

Debugging Tips

You must thoroughly debug your UDX before deploying it to a production environment. The following tips can help you get your UDX ready for deployment.

Use a Single Node For Initial Debugging

You can attach to the Vertica process using a debugger such as `gdb` to debug your UDX code. Doing this in a multi-node environment, however, is very difficult. Therefore, consider setting up a single-node Vertica test environment to initially debug your UDX.

Write Messages to the Vertica Log

You can write to log files using the `ServerInterface.log` function. Every function in your UDX receives an instance of the `ServerInterface` object, so you can call the log function from anywhere in your UDX. The function acts similarly to `printf`, taking a formatted string, and an optional set of values and writing the string to a log file. Where the message is written depends on whether your function runs in fenced mode or unfenced mode:

- Functions running in unfenced mode write their messages into the `vertica.log` file in the catalog directory.
- Functions running in fenced mode write their messages into a log file named `UDxLogs/UDxFencedProcesses.log` in the catalog directory.

To help identify your function's output, Vertica adds the SQL function name bound to your UDX function to the log message.

The following code fragment shows how you can add a call to `srvInterface.log` in the `Add2ints` example code's `processBlock()` function to log its input values:

```
const vint a = arg_reader.getIntRef(0);
const vint b = arg_reader.getIntRef(1);
srvInterface.log("got a: %d and b: %d", (int) a, (int) b);
```

This code generates an entries in the log file for each row the UDX processes. For example:

```
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 1 and b: 2
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 2 and b: 2
```

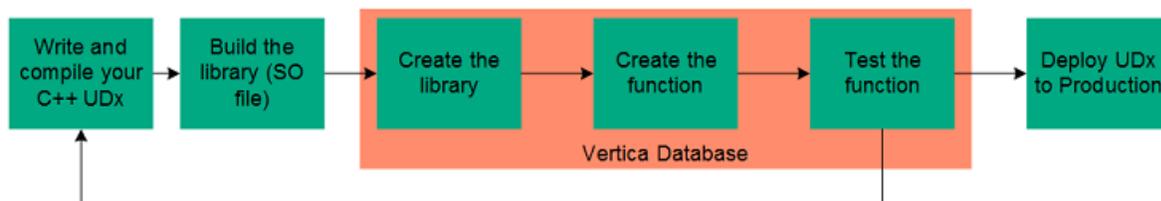
```
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 3 and b: 2
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 1 and b: 4
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 5 and b: 2
```

See [Monitoring the Log Files](#) in the Administrator's Guide for details on viewing the Vertica log files.

Developing with the C++ SDK

The Vertica SDK supports writing both fenced and unfenced UDxs of all types in C++ 11. You can download, compile, and run the examples; see [Downloading and Running UDx Example Code](#). Running the examples is a good way to verify that your development environment has all needed libraries.

If you do not have access to a Vertica test environment, you can install Vertica on your development machine and run a single node. Each time you rebuild your UDx library, you need to re-install it into Vertica. The following diagram illustrates the typical development cycle.



Extending Vertica provides a high-level description of the key APIs in the SDK. For more details, see the [C++ SDK Documentation](#).

Setting Up the C++ SDK

The Vertica C++ Software Development Kit (SDK) is distributed as part of the server installation. It contains the source and header files you need to create your UDx library. For examples that you can compile and run, see [Downloading and Running UDx Example Code](#). For requirements for your development environment, see [Setting Up a Development Environment](#).

The SDK files are located in the `sdk` subdirectory under the root Vertica server directory (usually, `/opt/vertica/sdk`). This directory contains a subdirectory, `include`, which contains the headers and source files needed to compile UDx libraries.

There are two files in the `include` directory you need when compiling your UDx:

- `Vertica.h` is the main header file for the SDK. Your UDX code needs to include this file in order to find the SDK's definitions.
- `Vertica.cpp` contains support code that needs to be compiled into the UDX library.

Much of the Vertica SDK API is defined in the `VerticaUDx.h` header file (which is included by the `Vertica.h` file). If you're curious, you might want to review the contents of this file in addition to reading the API documentation.

Finding the Current SDK Version

You must develop your UDX using the same SDK version as the database in which you plan to use it. To display the SDK version currently installed on your system, run the following command in `vsql`:

```
=> SELECT sdk_version();
```

Running the Examples

You can download the examples from the GitHub repository (see [Downloading and Running UDX Example Code](#)). Compiling and running the examples helps you to ensure that your development environment is properly set up.

To compile all of the examples, including the Java examples, issue the following command in the `Java-and-C++` directory under the examples directory:

```
$ make
```

Note: To compile the examples, you must have a `g++` development environment installed. To install a `g++` development environment on Red Hat systems, run `yum install gcc gcc-c++ make`.

Compiling Your C++ Library

GNU `g++` is the only supported compiler for compiling UDX libraries. Always compile your UDX code on the same version of Linux that you use on your Vertica cluster.

When compiling your library, you must always:

- Pass the `-shared` and `-fPIC` flags to the linker. The simplest method is to just pass these flags to `g++` when you compile and link your library.
- Use the `-Wno-unused-value` flag to suppress warnings when macro arguments are not used. If you do not use this flag, you may get "left-hand operand of comma has no effect" warnings.
- Compile `sdk/include/Vertica.cpp` and link it into your library. This file contains support routines that help your UDX communicate with Vertica. The easiest way to do this is to include it in the `g++` command to compile your library. Vertica supplies this file as C++ source rather than a library to limit library compatibility issues.
- Add the Vertica SDK include directory in the include search path using the `g++ -I` flag.

The SDK examples include a working makefile. See [Downloading and Running UDX Example Code](#).

Example of Compiling a UDX

The following command compiles a UDX contained in a single source file named `MyUDX.cpp` into a shared library named `MyUDX.so`:

```
g++ -I /opt/vertica/sdk/include -Wall -shared -Wno-unused-value \  
-fPIC -o MyUDX.so MyUDX.cpp /opt/vertica/sdk/include/Vertica.cpp
```

Important: Micro Focus only supports UDX development on 64-bit architectures.

After you debug your UDX, you are ready to deploy it. Recompile your UDX using the `-O3` flag to enable compiler optimization.

You can add additional source files to your library by adding them to the command line. You can also compile them separately and then link them together.

Tip: The examples subdirectory in the Vertica SDK directory contains a make file that you can use as starting point for your own UDX project.

Handling External Libraries

You must link your UDX library to any supporting libraries that your UDX code relies on. These libraries might be either ones you developed or others provided by third parties. You have two options for linking:

- Statically link the support libraries into your UDx. The benefit of this method is that your UDx library does not rely on external files. Having a single UDx library file simplifies deployment because you just transfer a single file to your Vertica cluster. This method's main drawback is that it increases the size of your UDx library file.
- Dynamically link the library to your UDx. You must sometimes use dynamic linking if a third-party library does not allow static linking. In this case, you must copy the libraries to your Vertica cluster in addition to your UDx library file.

Adding Metadata to C++ Libraries

You can add metadata, such as author name, the version of the library, a description of your library, and so on to your library. This metadata lets you track the version of your function that is deployed on an Vertica Analytics Platform cluster and lets third-party users of your function know who created the function. Your library's metadata appears in the [USER_LIBRARIES](#) system table after your library has been loaded into the Vertica Analytics Platform catalog.

You declare the metadata for your library by calling the `RegisterLibrary()` function in one of the source files for your UDx. If there is more than one function call in the source files for your UDx, whichever gets interpreted last as Vertica Analytics Platform loads the library is used to determine the library's metadata.

The `RegisterLibrary()` function takes eight string parameters:

```
RegisterLibrary(author,  
                library_build_tag,  
                library_version,  
                library_sdk_version,  
                source_url,  
                description,  
                licenses_required,  
                signature);
```

- `author` contains whatever name you want associated with the creation of the library (your own name or your company's name for example).
- `library_build_tag` is a string you want to use to represent the specific build of the library (for example, the SVN revision number or a timestamp of when the library was compiled). This is useful for tracking instances of your library as you are developing them.
- `library_version` is the version of your library. You can use whatever numbering or naming scheme you want.

- `library_sdk_version` is the version of the Vertica Analytics Platform SDK Library for which you've compiled the library.

Note: This field isn't used to determine whether a library is compatible with a version of the Vertica Analytics Platform server. The version of the Vertica Analytics Platform SDK you use to compile your library is embedded in the library when you compile it. It is this information that Vertica Analytics Platform server uses to determine if your library is compatible with it.

- `source_url` is a URL where users of your function can find more information about it. This can be your company's website, the GitHub page hosting your library's source code, or whatever site you like.
- `description` is a concise description of your library.
- `licenses_required` is a placeholder for licensing information. You must pass an empty string for this value.
- `signature` is a placeholder for a signature that will authenticate your library. You must pass an empty string for this value.

For example, the following code demonstrates adding metadata to the `Add2Ints` example (see [C++ Example: Add2Ints](#)).

```
// Include the top-level Vertica SDK file
#include "Vertica.h"
// Using the Vertica namespace means we don't have to prefix all
// class references with Vertica::
using namespace Vertica;
/*
 * ScalarFunction implementation for a UDSF that adds
 * two numbers together.
 */

class Add2Ints : public ScalarFunction
{
public:
    /*
     * This function does all of the actual processing for the UDX.
     * In this case, it simply reads two integer values and returns
     * their sum.
     *
     * The inputs are retrieved via arg_reader
     * The outputs are returned via arg_writer
     */
    virtual void processBlock(ServerInterface &srvInterface,
                              BlockReader &arg_reader,
                              BlockWriter &res_writer)
    {
```

```
// While we have input to process
do
{
    // Read the two integer input parameters by calling the
    // BlockReader.getIntRef class functionconst
    vint a = arg_reader.getIntRef(0);
    const vint b = arg_reader.getIntRef(1);
    // Call BlockWriter.setInt to store the output value, which is the
    // two input values added together
    res_writer.setInt(a+b);
    // Finish writing the row, and advance to the next output row
    res_writer.next();
    // Continue looping until there are no more input rows
}
while (arg_reader.next());
}
};

/*
 * This class provides metadata about the ScalarFunction class, and
 * also instantiates a member of that class when needed.
 */
class Add2IntsFactory : public ScalarFunctionFactory
{
    // return an instance of Add2Ints to perform the actual addition.
    virtual ScalarFunction *createScalarFunction(ServerInterface &interface)
    {
        // Calls the vt_createFuncObj to create the new Add2Ints class instance.
        return vt_createFuncObj(interface.allocator, Add2Ints);
    }

    // This function returns the description of the input and outputs of the
    // Add2Ints class's processBlock function. It stores this information in
    // two ColumnTypes objects, one for the input parameters, and one for
    // the return value.
    virtual void getPrototype(ServerInterface &interface,
                              ColumnTypes &argTypes,
                              ColumnTypes &returnType)
    {
        // Takes two ints as inputs, so add ints to the argTypes object
        argTypes.addInt();
        argTypes.addInt();
        // returns a single int, so add a single int to the returnType object.
        // Note that ScalarFunctions *always* return a single value.
        returnType.addInt();
    }
};

// Register the factory with Vertica
RegisterFactory(Add2IntsFactory);

// Register the library's metadata.
RegisterLibrary("Whizzo Analytics Ltd.",
               "1234",
               "2.0",
               "7.0.0",
               "http://www.example.com/add2ints",
               "Add 2 Integer Library",
               "",
               "");
```

Loading the library and querying the `USER_LIBRARIES` system table shows the metadata supplied in the call to `RegisterLibrary()`:

```
=> CREATE LIBRARY add2intslib AS '/home/dbadmin/add2ints.so';
CREATE LIBRARY
=> \x
Expanded display is on.
=> SELECT * FROM USER_LIBRARIES WHERE lib_name = 'add2intslib';
-[ RECORD 1 ]-----+-----
schema_name      | public
lib_name         | add2intslib
lib_oid          | 45035996273869808
author           | Whizzo Analytics Ltd.
owner_id         | 45035996273704962
lib_file_name    | public_add2intslib_45035996273869808.so
md5_sum          | 732c9e145d447c8ac6e7304313d3b8a0
sdk_version      | v7.0.0-20131105
revision         | 125200
lib_build_tag    | 1234
lib_version      | 2.0
lib_sdk_version  | 7.0.0
source_url       | http://www.example.com/add2ints
description      | Add 2 Integer Library
licenses_required |
signature        |
```

C++ SDK Data Types

The Vertica SDK has typedefs and classes for representing Vertica data types within your UDX code. Using these typedefs ensures data type compatibility between the data your UDX processes and generates and the Vertica database. The following table describes some of the typedefs available. Consult the VerticaC++ SDK Documentation for a complete list, as well as lists of helper functions to convert and manipulate these data types.

Type Definition	Description
Interval	A Vertica interval
IntervalYM	A Vertica year-to-month interval.
Timestamp	A Vertica timestamp
vint	A standard Vertica 64-bit integer
vint_null	A null value for integer values
vbool	A Boolean value in Vertica

Type Definition	Description
vbool_null	A null value for a Boolean data types
vfloat	A Vertica floating point value
VString	String data types (such as varchar and char) Note: Do not use a VString object to hold an intermediate result. Use a <code>std::string</code> or <code>char[]</code> instead.
VNumeric	Fixed-point data types from Vertica

Notes

- When making some Vertica SDK API calls (such as `VerticaType::getNumericLength()`) on objects, make sure they have the correct data type. To minimize overhead and improve performance, most of the APIs do not check the data types of the objects on which they are called. Calling a function on an incorrect data type can result in an error.
- A NULL Vertica value string data type is converted into an empty C++ string.
- You cannot create instances of `VString` or `VNumeric` yourself. You can manipulate the values of existing objects of these classes that Vertica passes to your UDX, and extract values from them. However, only Vertica can instantiate these classes.

Handling Errors

If your UDX encounters some sort of error, it can report it back to Vertica using the `vt_report_error` macro. When called, this macro halts the execution of the UDX and causes the statement that called the function to fail. The macro takes two parameters: an error number and an error message string. Both the error number and message appear in the error that Vertica reports to the user. The error number is not defined by Vertica. You can use whatever value that you wish.

For example, the following `ScalarFunction` class divides two integers. To prevent division by zero, it tests the second parameter. If it is zero, the function reports the error back to Vertica.

```
/*  
 * Demonstrate reporting an error  
 */
```

```
class Div2ints : public ScalarFunction
{
public:
    virtual void processBlock(ServerInterface &srvInterface,
                             BlockReader &arg_reader,
                             BlockWriter &res_writer)
    {
        // While we have inputs to process
        do
        {
            const vint a = arg_reader.getIntRef(0);
            const vint b = arg_reader.getIntRef(1);
            if (b == 0)
            {
                vt_report_error(1, "Attempted divide by zero");
            }
            res_writer.setInt(a/b);
            res_writer.next();
        }
        while (arg_reader.next());
    }
};
```

Loading and invoking the function demonstrates how the error appears to the user.

```
=> CREATE LIBRARY Div2IntsLib AS '/home/dbadmin/Div2ints.so';
CREATE LIBRARY
=> CREATE FUNCTION div2ints AS LANGUAGE 'C++' NAME 'Div2intsInfo' LIBRARY Div2IntsLib;
CREATE FUNCTION
=> SELECT div2ints(25, 5);
div2ints
-----
          5
(1 row)
=> SELECT * FROM MyTable;
 a | b
----+---
 12 | 6
  7 | 0
 12 | 2
 18 | 9
(4 rows)
=> SELECT * FROM MyTable WHERE div2ints(a, b) > 2;
ERROR:  Error in calling processBlock() for User Defined Scalar Function
div2ints at Div2ints.cpp:21, error code: 1, message: Attempted divide by zero
```

Your function must not allow an exception to be passed back to Vertica. You should use a top-level try-catch block to catch any stray exceptions that might be thrown by your code or any functions or libraries your code calls. This is especially important when running a UDX in unfenced mode. Any errors in an unfenced UDX can result in database instability or even data loss.

Handling Cancel Requests

You can cancel a query that calls your UDX (usually, by pressing CTRL+C in vsqll). How Vertica handles the cancellation of the query and your UDX depends on whether your UDX is running in fenced or unfenced mode:

- If your UDX is running in unfenced mode, Vertica either stops the function when it requests a new block of input or output, or waits until your function completes running and discards the results.
- If your UDX is running in [Fenced Mode](#), Vertica kills the zygote process that is running your function if it continues processing past a timeout.

See [Fenced Mode](#) for more information about running functions in fenced mode.

To give you more control over what happens to your function when the user cancels its query, the Vertica SDK includes an API for some UDXs to handle cancellation. Any function class that inherits from the `Vertica::UDXObjectCancelable` class can test whether the query calling it has been canceled using a function named `isCanceled()`. Your function can also implement a callback function named `cancel()` that Vertica calls when the function's query is canceled. Currently, the two classes that inherit from `UDXObjectCancelable` are `TransformFunction` and `AnalyticFunction`.

Exiting When the Calling Query Has Been Canceled

The `processPartition()` function in your User-Defined Transform Function (UDTF) or Analytic Function (UDAnF) can call `Vertica::UDXObjectCancelable.isCanceled()` to determine if the user has canceled the query that called it. If `isCanceled()` returns true, the query has been canceled and your `processPartition()` function should exit immediately to prevent it from wasting CPU time. If your UDX is not running in [Fenced Mode](#), Vertica cannot halt your function, and has to wait for it to finish. If it is running in fenced mode, Vertica can eventually kill the side process running it, but not until it has wasted some processing time.

How often your `processPartition()` function calls `isCanceled()` depends on how much processing it performs on each row of data. Calling `isCanceled()` does add some overhead to your function, so you shouldn't call it too often. For transforms that do not perform lengthy processing, you could check for cancellation every 100 or 1000 rows or so. If your `processPartition()` function performs extensive processing for each row, you may want to check `isCanceled()` every 10 or so rows.

The following code fragment shows how you could have the `StringTokenizer` UDTF example check whether its query has been canceled:

```
// The primary class for the StringTokenizer UDTF.
class StringTokenizer : public TransformFunction {
  // Called for each partition in the table. Receives the data from
  // The source table and
  virtual void processPartition(ServerInterface &srvInterface,
                               PartitionReader &inputReader,
                               PartitionWriter &outputWriter) {
    try {
      // Loop through the input rows
      int rowCount = 0; // Count the number of rows processed.
      do {
        rowCount++; // Processing a new row of data
        // Check for cancellation every 100 rows.
        if (rowCount % 100 == 0)
        {
          if (isCanceled()) // See if query has been canceled
          {
            // Log cancellation
            srvInterface.log("Got canceled!");
            return; // Exit out of UDTF immediately.
          }
        }
      }
      // Rest of the function here
    }
  }
}
```

This example checks for cancellation after processing 100 rows in the partition of data. If the query has been canceled, the example logs a message, then returns to the caller to exit the function.

Note: You need to strike a balance between adding overhead to your functions by calling `isCanceled()` and having your functions waste CPU time by running after their query has been canceled (usually, a rare event). For functions such as `StringTokenizer` which have a low overall processing cost, it usually does not make sense to test for cancellation. The cost of adding overhead to all function calls outweighs the amount of resources wasted by having the function run to completion or having its zygote process killed by Vertica on the rare occasions that its query is canceled.

Implementing the Cancel Callback Function

Your User-Defined Transform Function (UDTF) or Analytic Function (UDAF) can implement a `cancel()` callback function that Vertica calls if the query that called the function has been canceled. You usually implement this function to perform an orderly shutdown of any additional processing that your UDF spawned. For example, you can have your `cancel()` function shut down threads that your UDF has spawned or signal a third-party library that it needs to stop processing and exit. Your `cancel()` function should leave your UDF's function

class ready to be destroyed, since Vertica calls the UDX's destroy function after the cancel function has exited.

Notes

- If your UDTF or UDAF does not implement `cancel()`, Vertica assumes your UDX does not need to perform any special cancel processing, and calls the function class's `destroy()` function to have it free any resources (see [Resource Use for C++ UDXs](#)).
- Your `cancel()` function is called from a different thread than the thread running your UDX's `processPartition()` function.
- The call to the `cancel()` function is not synchronized in any way with your UDX's `processPartition()` function. If you need your `processPartition()` function to exit before your `cancel()` function performs some action (killing threads, for example) you need to have the two function synchronize their actions.
- If your `cancel()` function runs for too long, Vertica kills the side process running your function, if it is running in [Fenced Mode](#).

Resource Use for C++ UDxs

Your UDxs consume at least a small amount of memory by instantiating classes and creating local variables. This basic memory usage by UDxs is small enough that you do not need to be concerned about it.

If your UDx needs to allocate more than one or two megabytes of memory for data structures, or requires access to additional resources such as files, you must inform Vertica about its resource use. Vertica can then ensure that the resources your UDx requires are available before running a query that uses it. Even moderate memory use (10MB per invocation of a UDx, for example) can become an issue if there are many simultaneous queries that call it.

Note: If your UDx allocates its own memory, you must make **absolutely sure** it properly frees it. Failing to free even a single byte of allocated memory can have huge consequences if your UDx is called to operate on a multi-million-row table. Instead of having your code allocate its own memory, you should use the C++ `vt_alloc` macro, which uses Vertica's own memory manager to allocate and track memory. This memory is guaranteed to be properly disposed of when your UDx completes execution. See [Allocating Resources for UDxs](#) for more information.

Allocating Resources for UDxs

You have two options for allocating memory and file handles for your User-Defined Extensions (UDxs):

- Use Vertica SDK macros to allocate resources. This is the best method, since it uses Vertica's own resource manager, and guarantees that resources used by your UDx are reclaimed. See [Allocating Resources with the SDK Macros](#).
- Allocate resources in your UDxs yourself using standard C++ methods (instantiating objects using `new`, allocating memory blocks using `malloc()`, etc.). You must manually free these resources before your UDx exits.

Note: You must be extremely careful if you choose to allocate your own resources in your UDx. Failing to free resources properly will have significant negative impact, especially if your UDx is running in unfenced mode.

Whichever method you choose, you usually allocate resources in a function named `setup()` in your UDx class. This function is called after your UDx function object is instantiated, but before Vertica calls it to process data.

If you allocate memory on your own in the `setup()` function, you must free it in a corresponding function named `destroy()`. This function is called after your UDF has performed all of its processing. This function is also called if your UDF returns an error (see [Handling Errors](#)).

Note: Always use the `setup()` and `destroy()` functions to allocate and free resources instead of your own constructors and destructors. The memory for your UDF object is allocated from one of Vertica's own memory pools. Vertica always calls your UDF's `destroy()` function before it deallocates the object's memory. There is no guarantee that your UDF's destructor will be called before the object is deallocated. Using the `destroy()` function ensures that your UDF has a chance to free its allocated resources before it is destroyed.

The following code fragment demonstrates allocating and freeing memory using a `setup()` and `destroy()` function.

```
class MemoryAllocationExample : public ScalarFunction
{
public:
    uint64* myarray;
    // Called before running the UDF to allocate memory used throughout
    // the entire UDF processing.
    virtual void setup(ServerInterface &srvInterface, const SizedColumnTypes
        &argTypes)
    {
        try
        {
            // Allocate an array. This memory is directly allocated, rather than
            // letting Vertica do it. Remember to properly calculate the amount
            // of memory you need based on the data type you are allocating.
            // This example divides 500MB by 8, since that's the number of
            // bytes in a 64-bit unsigned integer.
            myarray = new uint64[1024 * 1024 * 500 / 8];
        }
        catch (std::bad_alloc &ba)
        {
            // Always check for exceptions caused by failed memory
            // allocations.
            vt_report_error(1, "Couldn't allocate memory :[%s]", ba.what());
        }
    }

    // Called after the UDF has processed all of its information. Use to free
    // any allocated resources.
    virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes
        &argTypes)
    {
        // srvInterface.log("RowNumber processed %d records", *count_ptr);
        try
        {
            // Properly dispose of the allocated memory.
            delete[] myarray;
        }
    }
}
```

```
catch (std::bad_alloc &ba)
{
    // Always check for exceptions caused by failed memory
    // allocations.
    vt_report_error(1, "Couldn't free memory :[%s]", ba.what());
}
}
```

Allocating Resources with the SDK Macros

The Vertica SDK provides three macros to allocate memory:

- `vt_alloc` allocates a block of memory to fit a specific data type (vint, struct, etc.).
- `vt_allocArray` allocates a block of memory to hold an array of a specific data type.
- `vt_allocSize` allocates an arbitrarily-sized block of memory.

All of these macros allocate their memory from memory pools managed by Vertica. The main benefit of allowing Vertica to manage your UDX's memory is that the memory is automatically reclaimed after your UDX has finished. This ensures there is no memory leaks in your UDX.

Because Vertica frees this memory automatically, do not attempt to free any of the memory you allocate through any of these macros. Attempting to free this memory results in run-time errors.

Informing Vertica of Resource Requirements

When you run your UDX in fenced mode, Vertica monitors its use of memory and file handles. If your UDX uses more than a few megabytes of memory or any file handles, it should tell Vertica about its resource requirements. Knowing the resource requirements of your UDX allows Vertica to determine whether it can run the UDX immediately or needs to queue the request until enough resources become available to run it.

Determining how much memory your UDX requires can be difficult in some cases. For example, if your UDX extracts unique data elements from a data set, there is potentially no bound on the number of data items. In this case, a useful technique is to run your UDX in a test environment and monitor its memory use on a node as it handles several differently-sized queries, then extrapolate its memory use based on the worst-case scenario it may face in your production environment. In all cases, it's usually a good idea to add a safety margin to the amount of memory you tell Vertica your UDX uses.

Note: The information on your UDX's resource needs that you pass to Vertica is used when planning the query execution. There is no way to change the amount of resources your UDX requests from Vertica while the UDX is actually running.

Your UDX informs Vertica of its resource needs by implementing the `getPerInstanceResources()` function in its factory class (see `Vertica::UDXFactory::getPerInstanceResources()` in the SDK documentation). If your UDX's factory class implements this function, Vertica calls it to determine the resources your UDX requires.

The `getPerInstanceResources()` function receives an instance of the `Vertica::VResources` struct. This struct contains fields that set the amount of memory and the number of file handles your UDX needs. By default, the Vertica server allocates zero bytes of memory and 100 file handles for each instance of your UDX.

Your implementation of the `getPerInstanceResources()` function sets the fields in the `VResources` struct based on the maximum resources your UDX may consume for each instance of the UDX function. So, if your UDX's `processBlock()` function creates a data structure that uses at most 100MB of memory, your UDX must set the `VResources.scratchMemory` field to at least 104857600 (the number of bytes in 100MB). Leave yourself a safety margin by increasing the number beyond what your UDX should normally consume. In this example, allocating 115000000 bytes (just under 110MB) is a good idea.

The following `ScalarFunctionFactory` class demonstrates calling `getPerInstanceResources()` to inform Vertica about the memory requirements of the `MemoryAllocationExample` class shown in [Allocating Resources for UDXs](#). It tells Vertica that the UDSF requires 510MB of memory (which is a bit more than the UDSF actually allocates, to be on the safe size).

```
class MemoryAllocationExampleFactory : public ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface
        &srvInterface)
    {
        return vt_createFuncObj(srvInterface.allocator, MemoryAllocationExample);
    }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
        Vertica::ColumnTypes &argTypes,
        Vertica::ColumnTypes &returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }
    // Tells Vertica the amount of resources that this UDF uses.
    virtual void getPerInstanceResources(ServerInterface &srvInterface,
        VResources &res)
    {
```

```
res.scratchMemory += 1024LL * 1024 * 510; // request 510MB of memory
}
};
```

Setting Memory Limits for Fenced-Mode UDxs

Vertica calls a fenced-mode UDx's implementation of `Vertica::UDXFactory::getPerInstanceResources()` to determine if there are enough free resources to run the query containing the UDx (see [Informing Vertica of Resource Requirements](#)). Since these reports are not generated by actual memory use, they can be inaccurate. Once started by Vertica, a UDx could allocate far more memory or file handles than it reported it needs.

The `FencedUDxMemoryLimitMB` configuration parameter lets you create an absolute memory limit for UDxs. Any attempt by a UDx to allocate more memory than this limit results in a `bad_alloc` exception. For more information on configuration parameters, see [Configuration Parameters](#) in the Administrator's Guide. For an example of setting `FencedUDxMemoryLimitMB`, see [How Resource Limits Are Enforced](#).

How Resource Limits Are Enforced

Before running a query, Vertica determines how much memory it requires to run. If the query contains a fenced-mode UDx which implements the `getPerInstanceResources()` function in its factory class, Vertica calls it to determine the amount of memory the UDx needs and adds this to the total required for the query. Based on these requirements, Vertica decides how to handle the query:

- If the total amount of memory required (including the amount that the UDxs report that they need) is larger than the session's `MEMORYCAP` or resource pool's `MAXMEMORYSIZE` setting, Vertica rejects the query. For more information about resource pools, see [Resource Pool Architecture](#) in the Administrator's Guide.
- If the amount of memory is below the limit set by the session and resource pool limits, but there is currently not enough free memory to run the query, Vertica queues it until enough resources become available.
- If there are enough free resources to run the query, Vertica executes it.

Note: Vertica has no other way to determine the amount of resources a UDx requires other than the values it reports using the `getPerInstanceResources()` function. A

UDx could use more resources than it claims, which could cause performance issues for other queries that are denied resources. You can set an absolute limit on the amount of memory UDxs can allocate. See [Setting Memory Limits for Fenced-Mode UDxs](#) for more information.

If the process executing your UDx attempts to allocate more memory than the limit set by the `FencedUDxMemoryLimitMB` configuration parameter, it receives a `bad_alloc` exception. For more information about `FencedUDxMemoryLimitMB`, see [Setting Memory Limits for Fenced-Mode UDxs](#).

Below is the output of loading a UDSF that consumes 500MB of memory, then changing the memory settings to cause out-of-memory errors. The `MemoryAllocationExample` UDSF in the following example is just the `Add2Ints` UDSF example altered as shown in [Allocating Resources for UDxs](#) and [Informing Vertica of Resource Requirements](#) to allocate 500MB of RAM.

```
=> CREATE LIBRARY mylib AS '/home/dbadmin/MemoryAllocationExample.so';
CREATE LIBRARY
=> CREATE FUNCTION usemem AS NAME 'MemoryAllocationExampleFactory' LIBRARY mylib
-> FENCED;
CREATE FUNCTION
=> SELECT usemem(1,2);
  usemem
-----
      3
(1 row)
```

The following statements demonstrate setting the session's `MEMORYCAP` to lower than the amount of memory that the UDSF reports it uses. This causes Vertica to return an error before it executes the UDSF.

```
=> SET SESSION MEMORYCAP '100M';
SET
=> SELECT usemem(1,2);
ERROR 3596: Insufficient resources to execute plan on pool sysquery
[Request exceeds session memory cap: 520328KB > 102400KB]
=> SET SESSION MEMORYCAP = default;
SET
```

The resource pool can also prevent a UDx from running if it requires more memory than is available in the pool. The following statements demonstrate the effect of creating and using a resource pool that has too little memory for the UDSF to run. Similar to the session's `MAXMEMORYCAP` limit, the pool's `MAXMEMORYSIZE` setting prevents Vertica from executing the query containing the UDSF.

```
=> CREATE RESOURCE POOL small MEMORYSIZE '100M' MAXMEMORYSIZE '100M';
CREATE RESOURCE POOL
=> SET SESSION RESOURCE POOL small;
SET
=> CREATE TABLE ExampleTable(a int, b int);
```

```
CREATE TABLE
=> INSERT /*+direct*/ INTO ExampleTable VALUES (1,2);
  OUTPUT
  -----
      1
(1 row)
=> SELECT usemem(a, b) FROM ExampleTable;
ERROR 3596: Insufficient resources to execute plan on pool small
[Request Too Large:Memory(KB) Exceeded: Requested = 523136, Free = 102400 (Limit = 102400, Used = 0)]
=> DROP RESOURCE POOL small; --Dropping the pool resets the session's pool
DROP RESOURCE POOL
```

Finally, setting the `FencedUDxMemoryLimitMB` configuration parameter to lower than the UDx actually allocates results in the UDx throwing an exception. This is a different case than either of the previous two examples, since the query actually executes. The UDx's code needs to catch and handle the exception. In this example, it uses the `vt_report_error` macro to report the error back to Vertica and exit.

```
=> ALTER DATABASE mydb SET FencedUDxMemoryLimitMB = 300;
=> SELECT usemem(1,2);
      ERROR 3412: Failure in UDx RPC call InvokeSetup(): Error calling setup() in
      User Defined Object [usemem] at [MemoryAllocationExample.cpp:32], error code:
      1, message: Couldn't allocate memory :[std::bad_alloc]
=> ALTER DATABASE mydb SET FencedUDxMemoryLimitMB = -1;
=> SELECT usemem(1,2);
  usemem
  -----
      3
(1 row)
```

See Also

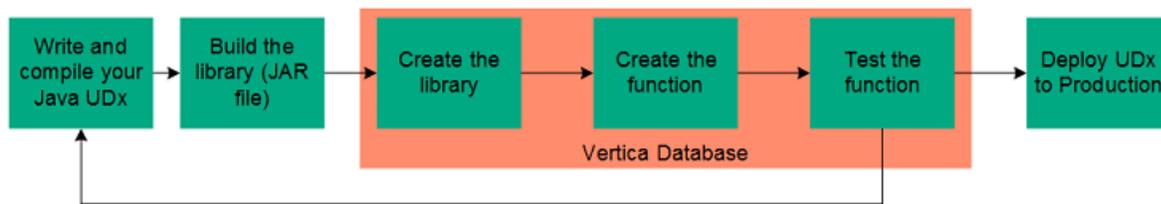
- [SET SESSION RESOURCE_POOL](#)
- [SET SESSION MEMORYCAP](#)
- [SET_CONFIG_PARAMETER](#)

Developing with the Java SDK

The Vertica SDK supports writing Java UDxs of all types except aggregate functions. All Java UDxs are fenced.

You can download, compile, and run the examples; see [Downloading and Running UDx Example Code](#). Running the examples is a good way to verify that your development environment has all needed libraries.

If you do not have access to a Vertica test environment, you can install Vertica on your development machine and run a single node. Each time you rebuild your UDX library, you need to re-install it into Vertica. The following diagram illustrates the typical development cycle.



Extending Vertica provides a high-level description of the key APIs in the SDK. For more details, see the [Java SDK Documentation](#).

Setting Up the Java SDK

The Vertica Java Software Development Kit (SDK) is distributed as part of the server installation. It contains the source and JAR files you need to create your UDX library. For examples that you can compile and run, see [Downloading and Running UDX Example Code](#). For requirements for your development environment, see [Setting Up a Development Environment](#).

To use the SDK you need two files from the Java support package:

- `/opt/vertica/bin/VerticaSDK.jar` contains the Vertica Java SDK and other supporting files.
- `/opt/vertica/sdk/BuildInfo.java` contains version information about the SDK. You must compile this file and include it within your Java UDX JAR files.

If you are not doing your development on a database node, you can copy these two files from one of the database nodes to your development system.

The `BuildInfo.java` and `VerticaSDK.jar` files that you use to compile your UDX must be from the same SDK version. Both files must also match the version of the SDK files on your Vertica hosts. Versioning is only an issue if you are not compiling your UDXs on a Vertica host. If you are compiling on a separate development system, always refresh your copies of these two files and recompile your UDXs just before deploying them.

Finding the Current SDK Version

You must develop your UDX using the same SDK version as the database in which you plan to use it. To display the SDK version currently installed on your system, run the following command in `vsq`:

```
=> SELECT sdk_version();
```

Compiling BuildInfo.java

You need to compile the `BuildInfo.java` file into a class file, so you can include it in your Java UDx JAR library. If you are using a Vertica node as a development system, you can either:

- Copy the `BuildInfo.java` file to another location on your host.
- If you have root privileges, compile the `BuildInfo.java` file in place. (Only the root user has privileges to write files to the `/opt/vertica/sdk` directory.)

Compile the file using the following command. Replace *path* with the path to the file and *output-directory* with the directory where you will compile your UDxs.

```
$ javac -classpath /opt/vertica/bin/VerticaSDK.jar \  
    /path/BuildInfo.java -d output-directory
```

If you use an IDE such as Eclipse, you can include the `BuildInfo.java` file in your project instead of compiling it separately. You must also add the `VerticaSDK.jar` file to the project's build path. See your IDE's documentation for details on how to include files and libraries in your projects.

Running the Examples

You can download the examples from the GitHub repository (see [Downloading and Running UDx Example Code](#)). Compiling and running the examples helps you to ensure that your development environment is properly set up.

If you have not already done so, set the `JAVA_HOME` environment variable to your JDK (not JRE) directory.

To compile all of the examples, including the Java examples, issue the following command in the `Java-and-C++` directory under the examples directory:

```
$ make
```

To compile only the Java examples, issue the following command in the `Java-and-C++` directory under the examples directory:

```
$ make JavaFunctions
```

Note: To compile the examples, you must have make installed. To install make on Red Hat systems, run `yum install make`.

Compiling and Packaging a Java Library

Before you can use your Java UDX, you need to compile it and package it into a JAR file.

The SDK examples include a working makefile. See [Downloading and Running UDX Example Code](#).

Compile Your Java UDX

You must include the SDK JAR file in the classpath when you compile your Java UDX source files so the Java compiler can resolve the Vertica API calls. If you are using the command-line Java compiler on a host in your database cluster, enter this command:

```
$ javac -classpath /opt/vertica/bin/VerticaSDK.jar factorySource.java \  
[functionSource.java...] -d output-directory
```

If all of your source files are in the same directory, you can use `*.java` on the command line instead of listing the files individually.

If you are using an IDE, verify that a copy of the `VerticaSDK.jar` file is in the build path.

UDX Class File Organization

After you compile your UDX, you must package its class files and the `BuildInfo.class` file into a JAR file.

Note: You can package as many UDXs as you want into the same JAR file. Bundling your UDXs together saves you from having to load multiple libraries.

To use the `jar` command packaged as part of the JDK, you must organize your UDX class files into a directory structure matching your class package structure. For example, suppose your UDX's factory class has a fully-qualified name of `com.mycompany.udfs.Add2ints`. In this case, your class files must be in the directory hierarchy `com/mycompany/udfs` relative to your project's base directory. In addition, you must have a copy of the `BuildInfo.class` file in the path `com/vertica/sdk` so that it can be included in the JAR file. This class must appear in your JAR file to indicate the SDK version that was used to compile your Java UDX.

The JAR file for the Add2ints UDSF example explained in [Java Example: Add2Ints](#) has the following directory structure after compilation:

```
com/vertica/sdk/BuildInfo.class
com/mycompany/example/Add2intsFactory.class
com/mycompany/example/Add2intsFactory$Add2ints.class
```

Package Your UDx Into a JAR File

To create a JAR file from the command line:

1. Change to the root directory of your project.
2. Use the `jar` command to package the `BuildInfo.class` file and all of the classes in your UDx:

```
# jar -cvf libname.jar com/vertica/sdk/BuildInfo.class \  
    packagePath/*.class
```

When you type this command, *libname* is the filename you have chosen for your JAR file (choose whatever name you like), and *packagePath* is the path to the directory containing your UDx's class files.

- For example, to package the files from the Add2ints example, you use the command:

```
# jar -cvf Add2intsLib.jar com/vertica/sdk/BuildInfo.class \  
    com/mycompany/example/*.class
```

- More simply, if you compiled `BuildInfo.class` and your class files into the same root directory, you can use the following command:

```
# jar -cvf Add2intsLib.jar .
```

You must include all of the class files that make up your UDx in your JAR file. Your UDx always consists of at least two classes (the factory class and the function class). Even if you defined your function class as an inner class of your factory class, Java generates a separate class file for the inner class.

After you package your UDx into a JAR file, you are ready to deploy it to your Vertica database.

Handling Java UDX Dependencies

If your Java UDX relies on one or more external libraries, you can handle the dependencies in one of three ways:

- Bundle the JAR files into your UDX JAR file using a tool such as [One-JAR](#) or Eclipse Runnable JAR Export Wizard.
- Unpack the JAR file and then repack its contents in your UDX's JAR file.
- Copy the libraries to your Vertica cluster in addition to your UDX library. Then, use the `DEPENDS` keyword of the [CREATE LIBRARY](#) statement to tell Vertica that the UDX library depends on the external libraries. This keyword acts as a library-specific `CLASSPATH` setting. Vertica distributes the support libraries to all of the nodes in the cluster and sets the class path for the UDX so it can find them.

If your UDX depends on native libraries (SO files), use the `DEPENDS` keyword to specify their path. When you call `System.loadLibrary` in your UDX (which you must do before using a native library), this function uses the `DEPENDS` path to find them. You do not need to also set the `LD_LIBRARY_PATH` environment variable.

Important: In previous versions of Vertica, the `JavaClassPathForUDx` configuration parameter set the locations of external libraries for all Java UDXs. In Vertica Version 7.1, this configuration parameter is deprecated. It will be removed in a future version. If you upgrade from a previous version of Vertica, switch any existing Java UDXs that rely on support libraries to use the new `DEPENDS` keyword. To make this switch, use the [ALTER LIBRARY](#) statement to recreate the UDX library. In this statement, supply the `DEPENDS` keyword to specify the directory containing the support libraries.

External Library Example

The following example demonstrates using an external library with a Java UDX.

The following sample code defines a simple class, named `VowelRemover`. It contains a single method, named `removevowels`, that removes all of the vowels (the letters *a*, *e*, *i*, *o*, *u*, and *y*) from a string.

```
package com.mycompany.libs;  
  
public class VowelRemover {
```

```
public String removevowels(String input) {
    return input.replaceAll("(?i)[aeiouy]", "");
}
};
```

You can compile this class and package it into a JAR file with the following commands:

```
$ javac -g com/mycompany/libs/VowelRemover.java
$ jar cf mycompanylibs.jar com/mycompany/libs/VowelRemover.class
```

The following code defines a Java UDSF, named `DeleteVowels`, that uses the library defined in the preceding example code. `DeleteVowels` accepts a single `VARCHAR` as input, and returns a `VARCHAR`.

```
package com.mycompany.udx;
// Import the support class created earlier
import com.mycompany.libs.VowelRemover;
// Import the Vertica SDK
import com.vertica.sdk.*;

public class DeleteVowelsFactory extends ScalarFunctionFactory {

    @Override
    public ScalarFunction createScalarFunction(ServerInterface arg0) {
        return new DeleteVowels();
    }

    @Override
    public void getPrototype(ServerInterface arg0, ColumnTypes argTypes,
        ColumnTypes returnTypes) {
        // Accept a single string and return a single string.
        argTypes.addVarchar();
        returnTypes.addVarchar();
    }

    @Override
    public void getReturnType(ServerInterface srvInterface,
        SizedColumnTypes argTypes,
        SizedColumnTypes returnType){
        returnType.addVarchar(
            // Output will be no larger than the input.
            argTypes.getColumnType(0).getStringLength(), "RemovedVowels");
    }

    public class DeleteVowels extends ScalarFunction
    {
        @Override
        public void processBlock(ServerInterface arg0, BlockReader argReader,
            BlockWriter resWriter) throws UdfException, DestroyInvocation {

            // Create an instance of the VowelRemover object defined in
            // the library.
            VowelRemover remover = new VowelRemover();

            do {
                String instr = argReader.getString(0);
```

```
    // Call the removevowels method defined in the library.
    resWriter.setString(remover.removevowels(instr));
    resWriter.next();
  } while (argReader.next());
}
}
```

Use the following commands to build the example UDSF and package it into a JAR:

- The first `javac` command compiles the SDK's `BuildInfo` class. Vertica requires all UDX libraries to contain this class. The `javac` command's `-d` option outputs the class file in the directory structure of your UDSF's source.
- The second `javac` command compiles the UDSF class. It adds the previously-created `mycompanylibs.jar` file to the class path so compiler can find the `VowelRemover` class.
- The `jar` command packages the `BuildInfo` and the classes for the UDX library together.

```
$ javac -g -cp /opt/vertica/bin/VerticaSDK.jar\
/opt/vertica/sdk/com/vertica/sdk/BuildInfo.java -d .
$ javac -g -cp mycompanylibs.jar:/opt/vertica/bin/VerticaSDK.jar\
com/mycompany/udx/DeleteVowelsFactory.java
$ jar cf DeleteVowelsLib.jar com/mycompany/udx/*.class \
com/vertica/sdk/*.class
```

To install the UDX library, you must copy both of the JAR files to a node in the Vertica cluster. Then, connect to the node to execute the `CREATE LIBRARY` statement.

The following example demonstrates how to load the UDX library after you copy the JAR files to the home directory of the `dbadmin` user. The `DEPENDS` keyword tells Vertica that the UDX library depends on the `mycompanylibs.jar` file.

```
=> CREATE LIBRARY DeleteVowelsLib AS
    '/home/dbadmin/DeleteVowelsLib.jar' DEPENDS '/home/dbadmin/mycompanylibs.jar'
    LANGUAGE 'JAVA';
CREATE LIBRARY
=> CREATE FUNCTION deleteVowels AS language 'java' NAME
    'com.mycompany.udx.DeleteVowelsFactory' LIBRARY DeleteVowelsLib;
CREATE FUNCTION
=> SELECT deleteVowels('I hate vowels!');
    deleteVowels
-----
    ht vwls!
(1 row)
```

Java and Vertica Data Types

The Vertica Java SDK converts Vertica's native data types into the appropriate Java data type. The following table lists the Vertica data types and their corresponding Java data types.

Vertica Data Type	Java Data Type
INTEGER	long
FLOAT	double
NUMERIC	com.vertica.sdk.VNumeric
DATE	java.sql.Date
CHAR, VARCHAR, LONG VARCHAR	com.vertica.sdk.VString
BINARY, VARBINARY, LONG VARBINARY	com.vertica.sdk.VString
TIMESTAMP	java.sql.Timestamp

Note: Some Vertica data types are not supported.

Setting BINARY, VARBINARY, and LONG VARBINARY Values

The Vertica BINARY, VARBINARY, and LONG VARBINARY data types are converted as the Java UDX SDK 's VString class. You can also set the value of a column with one of these data types with a ByteBuffer object (or a byte array wrapped in a ByteBuffer) using the PartitionWriter.setStringBytes() method. See the Java API UDX entry for PartitionWriter.setStringBytes() for more information.

Timestamps and Time Zones

When the SDK converts a Vertica timestamp into a Java timestamp, it uses the time zone of the JVM. If the JVM is running in a different time zone than the one used by Vertica, the results can be confusing.

Vertica stores timestamps in the database in UTC. (If a database time zone is set, the conversion is done at query time.) To prevent errors from the JVM time zone, add the following code to the processing method of your UDX:

```
TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
```

Strings

The Java SDK contains a class named `StringUtils` that assists you when manipulating string data. One of its more useful features is its `getStringBytes()` method. This method extracts bytes from a `String` in a way that prevents the creation of invalid strings. If you attempt to extract a substring that would split part of a multi-byte UTF-8 character, `getStringBytes()` truncates it to the nearest whole character.

Handling NULL Values

Your UDXs must be prepared to handle NULL values. These values usually must be handled separately from regular values.

Reading NULL Values

Your UDX reads data from instances of the `BlockReader` or `PartitionReader` classes. If the value of a column is NULL, the methods you use to get data (such as `getLong`) return a Java `null` reference. If you attempt to use the value without checking for NULL, the Java runtime will throw a null pointer exception.

You can test for null values before reading columns by using the data-type-specific methods (such as `isLongNull`, `isDoubleNull`, and `isBooleanNull`). For example, to test whether the `INTEGER` first column of your UDX's input is a NULL, you would use the statement:

```
// See if the Long value in column 0 is a NULL
if (inputReader.isLongNull(0)) {
    // value is null
    ...
}
```

Writing NULL Values

You output NULL values using type-specific methods on the `BlockWriter` and `PartitionWriter` classes (such as `setLongNull` and `setStringNull`). These methods take the column number to receive the NULL value. In addition, the `PartitionWriter` class has data-type specific set value methods (such as `setLongValue` and `setStringValue`). If you pass these methods a value, they set the output column to that value. If you pass them a Java `null` reference, they set the output column to NULL.

Handling Errors

If your UDX encounters an unrecoverable error, it should instantiate and throw a `UdfException`. The exception causes the transaction containing the function call to be rolled back.

The `UdfException` constructor takes a numeric code (which can be anything you want since it is just reported in the error message) and an error message string. If you want to report additional diagnostic information about the error, you can write messages to a log file before throwing the exception (see [Writing Messages to the Log File](#)).

The following code fragment demonstrates adding error checking to the `Add2ints` UDSF example (shown in [Java Example: Add2Ints](#)). If either of the arguments is `NULL`, the `processBlock()` method throws an exception.

```
@Override
public void processBlock(ServerInterface srvInterface,
                        BlockReader argReader,
                        BlockWriter resWriter)
    throws UdfException, DestroyInvocation
{
    do {
        // Test for NULL value. Throw exception if one occurs.
        if (argReader.isLongNull(0) || argReader.isLongNull(1)) {
            // No nulls allowed. Throw exception
            throw new UdfException(1234, "Cannot add a NULL value");
        }
    }
}
```

Note: This example isn't realistic, since you would likely just replace the `NULL` value with a zero or return a `NULL` value. Your UDX should only throw an exception if there is no way to compensate for the error.

When your UDX throws an exception, the side process running your UDX reports the error back to Vertica and exits. Vertica displays the error message contained in the exception and a stack trace to the user:

```
=> SELECT add2ints(2, NULL);
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error in User Defined Object [add2ints],
error code: 1234
com.vertica.sdk.UdfException: Cannot add a NULL value
    at com.mycompany.example.Add2intsFactory$Add2ints.processBlock(Add2intsFactory.java:37)
    at com.vertica.udxfence.UDXExecContext.processBlock(UDXExecContext.java:700)
    at com.vertica.udxfence.UDXExecContext.run(UDXExecContext.java:173)
    at java.lang.Thread.run(Thread.java:662)
```

Writing Messages to the Log File

Writing messages to a log is useful when you are debugging your Java UDXs, or you want to output additional information about an error condition. You can write messages to a log file by calling the `ServerInterface.log()` method, passing it a `printf()`-style `String` value along with any variables referenced in the string. (See the [java.util.Formatter class documentation](#) for details of formatting this string value.) An instance of the `ServerInterface` class is passed to the main processing method of every SDK class you can override.

The following code fragment demonstrates how you could log the values passed into the `Add2ints` UDSF example.

```
@Override
public void processBlock(ServerInterface srvInterface,
    BlockReader argReader,
    BlockWriter resWriter)
    throws UdfException, DestroyInvocation
{
    do {
        // Get the two integer arguments from the BlockReader
        long a = argReader.getLong(0);
        long b = argReader.getLong(1);
        // Log the input values
        srvInterface.log("Got values a=%d and b=%d", a, b);
    }
}
```

The messages are written to a log file stored in the catalog directory's `UDxLog` subdirectory named `UDxFencedProcessesJava.log`:

```
$ tail VMart/v_vmart_node0001_catalog/UDxLogs/UDxFencedProcesses.log
2012-12-12 10:23:47.649 [Java-2164] 0x01 UDx side process (Java) started
2012-12-12 10:23:47.871 [Java-2164] 0x0b [UserMessage] add2ints - Got
values a=5 and b=6
2012-12-12 10:23:48.598 [Java-2164] 0x0c Exiting UDx side process
```

The SQL name of the UDX is added to the log message, along with the string `[UserMessage]` to mark the entry as a message added by a call to the `log()` method. These additions make it easier for you to filter the log to find the messages generated by your UDX.

Handling Cancel Requests

The query that calls your UDX can be canceled (usually, by the user pressing CTRL+C in vsqll). When the calling query is canceled, Vertica begins a process of shutting down your UDX. Since UDTFs can perform lengthy and costly processing, the Vertica Java SDK defines several ways that Vertica attempts to signal UDTFs to terminate before it takes the step of killing the fenced-mode JVM process that is executing the UDTF. These attempts to signal the UDTF can help reduce the amount of CPU and memory that is wasted by having the UDX process continue processing after its results are no longer required.

When the user cancels a UDX, Vertica takes the following steps:

1. It sets the `isCanceled` property on UDTFs to true. Your `processPartition` methods can test this property to see if the function call has been canceled.
2. It calls UDTF's `TransformFunction.cancel` method. You should override this method to perform any shutdown tasks (such as killing threads).
3. It calls all types of UDX's `destroy` method. You should implement this method to free any resources your UDX has allocated.
4. It kills the JVM process running your UDX.

The topics in this section explain how your UDTF can use the cancel API.

Exiting When the Calling Query Has Been Canceled

Since User-Defined Transform Functions (UDTFs) often perform lengthy and CPU-intensive processing, it makes sense for them to terminate if the query that called them has been canceled. Exiting when the query has been canceled helps prevent wasting CPU cycles and memory on continued processing.

The `TransformFunction` class has a getter named `.isCanceled` that returns true if the calling query has been canceled. Your `processPartition` method can periodically check the value of this getter to determine if the query has been canceled, and exit if it has.

How often your `processPartition` function calls `isCanceled` depends on how much processing it performs on each row of data. Calling `isCanceled` does add overhead to your function, so you shouldn't call it too often. For transforms that do not perform lengthy processing, you could check for cancellation every 100 or 1000 rows. If your `processPartition` performs extensive processing for each row, you may want to check `isCanceled` every 10 or so rows.

The following code fragment shows how you could have the `StringTokenizer` UDTF example check whether its query has been canceled:

```
public class CancelableTokenizeString extends TransformFunction
{
    @Override
    public void processPartition(ServerInterface srvInterface,
        PartitionReader inputReader,
        PartitionWriter outputWriter)
        throws UdfException, DestroyInvocation
    {
        // Loop over all rows passed in in this partition.

        int rowcount = 0; // maintain count of rows processed
        do {
            rowcount++; // Processing new row

            // Check for cancelation every 100 rows
            if (rowcount % 100 == 0) {
                // Check to see if Vertica marked this class as canceled
                if (this.isCanceled()) {
                    srvInterface.log("Got canceled! Exiting...");
                    return;
                }
            }
        }
        // Rest of the function here
    }
}
```

This example checks for cancelation after processing 100 rows in the partition of data. If the query has been canceled, the example logs a message, then returns to the caller to exit the function.

Note: You need to strike a balance between adding overhead to your functions by calling `isCanceled` and having your functions waste CPU time by running after their query has been canceled (a rare event). For functions such as `StringTokenizer` which have a low overall processing cost, it usually does not make sense to test for cancelation. The cost of adding overhead to all function calls outweigh the amount of resources wasted by having the function run to completion or having its JVM process killed by Vertica on the rare occasions that its query is canceled.

Overriding the Cancel Method

Your User-Defined Transform Function (UDTF) can override the `TransformFunction.cancel` method that Vertica calls if the query that called the function has been canceled. You should override this method to perform an orderly shutdown of any additional processing that your UDF spawned. For example, you can have your `cancel` method shut down threads that your UDTF has spawned or signal a third-party library that it needs to stop processing and exit. Your `cancel` method must leave your UDTF's function class ready to

be destroyed, since Vertica calls the UDX's `destroy` method after the `cancel` method has exited.

Notes

- If your UDTF does not override `cancel`, Vertica assumes your UDTF does not need to perform any special cancel processing and calls the function class's `destroy` method to have it free any resources.
- Your `cancel` method is called from a different thread than the thread running your UDX's `processPartition` function.
- The call to the `cancel` method is not synchronized in any way with your UDTF's `processPartition` method. If you need your `processPartition` function to exit before your `cancel` method performs some action (killing threads, for example) you need to have the two methods synchronize their actions.
- If your `cancel` method runs for too long, Vertica kills the JVM side process your UDX.

Adding Metadata to Java UDX Libraries

You can add metadata, such as author name, the version of the library, a description of your library, and so on to your library. This metadata lets you track the version of your function that is deployed on an Vertica Analytics Platform cluster and lets third-party users of your function know who created the function. Your library's metadata appears in the [USER_LIBRARIES](#) system table after your library has been loaded into the Vertica Analytics Platform catalog.

To add metadata to your Java UDX library, you create a subclass of the `UDXLibrary` class that contains your library's metadata. You then include this class within your JAR file. When you load your class into the Vertica Analytics Platform catalog using the [CREATE LIBRARY](#) statement, looks for a subclass of `UDXLibrary` for the library's metadata.

In your subclass of `UDXLibrary`, you need to implement eight getters that return String values containing the library's metadata. The getters in this class are:

- `getAuthor()` returns the name you want associated with the creation of the library (your own name or your company's name for example).
- `getLibraryBuildTag()` returns whatever String you want to use to represent the specific build of the library (for example, the SVN revision number or a timestamp of when

the library was compiled). This is useful for tracking instances of your library as you are developing them.

- `getLibraryVersion()` returns the version of your library. You can use whatever numbering or naming scheme you want.
- `getLibrarySDKVersion()` returns the version of the Vertica Analytics Platform SDK Library for which you've compiled the library.

Note: This field isn't used to determine whether a library is compatible with a version of the Vertica Analytics Platform server. The version of the Vertica Analytics Platform SDK you use to compile your library is embedded in the library when you compile it. It is this information that Vertica Analytics Platform server uses to determine if your library is compatible with it.

- `getSourceUrl()` returns a URL where users of your function can find more information about it. This can be your company's website, the GitHub page hosting your library's source code, or whatever site you like.
- `getDescription()` returns a concise description of your library.
- `getLicensesRequired()` returns a placeholder for licensing information. You must pass an empty string for this value.
- `getSignature()` returns a placeholder for a signature that will authenticate your library. You must pass an empty string for this value.

For example, the following code demonstrates creating a `UDXLibrary` subclass to be included in the `Add2Ints` UDSF example JAR file (see [Java Example: Add2Ints](#)).

```
// Import the UDXLibrary class to hold the metadata
import com.vertica.sdk.UDXLibrary;

public class Add2IntsLibrary extends UDXLibrary
{
    // Return values for the metadata about this library.

    @Override public String getAuthor() {return "Whizzo Analytics Ltd.";}
    @Override public String getLibraryBuildTag() {return "1234";}
    @Override public String getLibraryVersion() {return "1.0";}
    @Override public String getLibrarySDKVersion() {return "7.0.0";}
    @Override public String getSourceUrl() {
        return "http://example.com/add2ints";
    }
    @Override public String getDescription() {
        return "My Awesome Add 2 Ints Library";
    }
}
```

```
@Override public String getLicensesRequired() {return "";}  
@Override public String getSignature() {return "";}  
}
```

When the library containing the `Add2IntsLibrary` class loaded, the metadata appears in the `USER_LIBRARIES` system table:

```
=> CREATE LIBRARY JavaAdd2IntsLib AS :libfile LANGUAGE 'JAVA';  
CREATE LIBRARY  
=> CREATE FUNCTION JavaAdd2Ints as LANGUAGE 'JAVA' name 'com.mycompany.example.Add2IntsFactory'  
library JavaAdd2IntsLib;  
CREATE FUNCTION  
=> \x  
Expanded display is on.  
=> SELECT * FROM USER_LIBRARIES WHERE lib_name = 'JavaAdd2IntsLib';  
-[ RECORD 1 ]-----+-----  
schema_name      | public  
lib_name         | JavaAdd2IntsLib  
lib_oid          | 45035996273869844  
author           | Whizzo Analytics Ltd.  
owner_id         | 45035996273704962  
lib_file_name    | public_JavaAdd2IntsLib_45035996273869844.jar  
md5_sum          | f3bfc76791daee95e4e2c0f8a8d2737f  
sdk_version      | v7.0.0-20131105  
revision         | 125200  
lib_build_tag    | 1234  
lib_version      | 1.0  
lib_sdk_version  | 7.0.0  
source_url       | http://example.com/add2ints  
description      | My Awesome Add 2 Ints Library  
licenses_required |  
signature        |
```

Java UDX Resource Management

Java Virtual Machines (JVMs) allocate a set amount of memory when they start. This set memory allocation complicates memory management for Java UDXs, because memory cannot be dynamically allocated and freed by the UDX as it is processing data. This differs from C++ UDXs which can dynamically allocate resources.

To control the amount of memory consumed by Java UDXs, Vertica has a memory pool named `jvm` that it uses to allocate memory for JVMs. If this memory pool is exhausted, queries that call Java UDXs block until enough memory in the pool becomes free to start a new JVM.

By default, the `jvm` pool has:

- no memory of its own assigned to it, so it borrows memory from the `GENERAL` pool.
- its `MAXMEMORYSIZE` set to either 10% of system memory or 2GB, whichever is smaller.

- its `PLANNEDCONCURRENCY` set to `AUTO`, so that it inherits the `GENERAL` pool's `PLANNEDCONCURRENCY` setting.

You can view the current settings for the `jvm` pool by querying the `RESOURCE_POOLS` table:

```
=> SELECT MAXMEMORYSIZE, PLANNEDCONCURRENCY FROM V_CATALOG.RESOURCE_POOLS WHERE NAME = 'jvm';
MAXMEMORYSIZE | PLANNEDCONCURRENCY
-----+-----
10%           | AUTO
```

When a SQL statement calls a Java UDX, Vertica checks if the `jvm` memory pool has enough memory to start a new JVM instance to execute the function call. Vertica starts each new JVM with its heap memory size set to approximately the `jvm` pool's `MAXMEMORYSIZE` parameter divided by its `PLANNEDCONCURRENCY` parameter. If the memory pool does not contain enough memory, the query blocks until another JVM exits and return their memory to the pool.

If your Java UDX attempts to consume more memory than has been allocated to the JVM's heap size, it exits with a memory error. You can attempt to resolve this issue by:

- increasing the `jvm` pool's `MAXMEMORYSIZE` parameter.
- decreasing the `jvm` pool's `PLANNEDCONCURRENCY` parameter.
- changing your Java UDX's code to consume less memory.

Adjusting the `jvm` Pool

When adjusting the `jvm` pool to your needs, you must consider two factors:

- the amount of RAM your Java UDX requires to run
- how many concurrent Java UDX functions you expect your database to run

You can learn the amount of memory your Java UDX needs using several methods. For example, your code can use Java's `Runtime` class to get an estimate of the total memory it has allocated and then log the value using `ServerInterface.log()`. (An instance of this class is passed to your UDX.) If you have multiple Java UDXs in your database, set the `jvm` pool memory size based on the UDX that uses the most memory.

The number of concurrent sessions that need to run Java UDXs may not be the same as the global `PLANNEDCONCURRENCY` setting. For example, you may have just a single user who runs a Java UDX, which means you can lower the `jvm` pool's `PLANNEDCONCURRENCY` setting to 1.

When you have an estimate for the amount of RAM and the number of concurrent user sessions that need to run Java UDXs, you can adjust the `jvm` pool to an appropriate size. Set the pool's `MAXMEMORYSIZE` to the maximum amount of RAM needed by the most demanding Java UDX multiplied by the number of concurrent user sessions that need to run Java UDXs. Set the pool's `PLANNEDCONCURRENCY` to the number of simultaneous user sessions that need to run Java UDXs.

For example, suppose your Java UDX requires up to 4GB of memory to run and you expect up to two user sessions use Java UDX's. You would use the following command to adjust the `jvm` pool:

```
=> ALTER RESOURCE POOL jvm MAXMEMORYSIZE '8G' PLANNEDCONCURRENCY 2;
```

The `MEMORYSIZE` is set to 8GB, which is the 4GB maximum memory use by the Java UDX multiplied by the 2 concurrent user sessions.

Note: The `PLANNEDCONCURRENCY` value is **not** the number of calls to Java UDX that you expect to happen simultaneously. Instead, it is the number of concurrently open user sessions that call Java UDXs at any time during the session. See below for more information.

See [Managing Workloads](#) in the Administrator's Guide for more information on tuning the `jvm` and other resource pools.

Freeing JVM Memory

The first time users call a Java UDX during their session, Vertica allocates memory from the `jvm` pool and starts a new JVM. This JVM remains running for as long as the user's session is open so it can process other Java UDX calls. Keeping the JVM running lowers the overhead of executing multiple Java UDXs by the same session. If the JVM did not remain open, each call to a Java UDX would require additional time for Vertica to allocate resources and start a new JVM. However, having the JVM remain open means that the JVM's memory remains allocated for the life of the session whether or not it will be used again.

If the `jvm` memory pool is depleted, queries containing Java UDXs either block until memory becomes available or eventually fail due a lack of resources. If you find queries blocking or failing for this reason, you can allocate more memory to the `jvm` pool and increase its `PLANNEDCONCURRENCY`. Another option is to ask users to call the [RELEASE_JVM_MEMORY](#) function when they no longer need to run Java UDXs. This function closes any JVM belonging to the user's session and returns its allocated memory to the `jvm` memory pool.

The following example demonstrates querying V_MONITOR.SESSIONS to find the memory allocated to JVMs by all sessions. It also demonstrates how the memory is allocated by a call to a Java UDX, and then freed by calling RELEASE_JVM_MEMORY.

```
=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
  user_name | external_memory_kb
-----+-----
  dbadmin   |          0
(1 row)

=> -- Call a Java UDX
=> SELECT add2ints(123,456);
  add2ints
-----
      579
(1 row)
=> -- JVM is now running and memory is allocated to it.
=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
  USER_NAME | EXTERNAL_MEMORY_KB
-----+-----
  dbadmin   |          79705
(1 row)

=> -- Shut down the JVM and deallocate memory
=> SELECT RELEASE_JVM_MEMORY();
          RELEASE_JVM_MEMORY
-----
Java process killed and memory released
(1 row)

=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
  USER_NAME | EXTERNAL_MEMORY_KB
-----+-----
  dbadmin   |          0
(1 row)
```

In rare cases, you may need to close all JVMs. For example, you may need to free memory for an important query, or several instances of a Java UDX may be taking too long to complete. You can use the [RELEASE_ALL_JVM_MEMORY](#) to close all of the JVMs in all user sessions:

```
=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
  USER_NAME | EXTERNAL_MEMORY_KB
-----+-----
  ExampleUser |          79705
  dbadmin     |          79705
(2 rows)

=> SELECT RELEASE_ALL_JVM_MEMORY();
          RELEASE_ALL_JVM_MEMORY
-----
Close all JVM sessions command sent. Check v_monitor.sessions for progress.
(1 row)

=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
  USER_NAME | EXTERNAL_MEMORY_KB
-----+-----
  dbadmin   |          0
```

(1 row)

Caution: This function terminates all JVMs, including ones that are currently executing Java UDxs. This will cause any query that is currently executing a Java UDX to return an error.

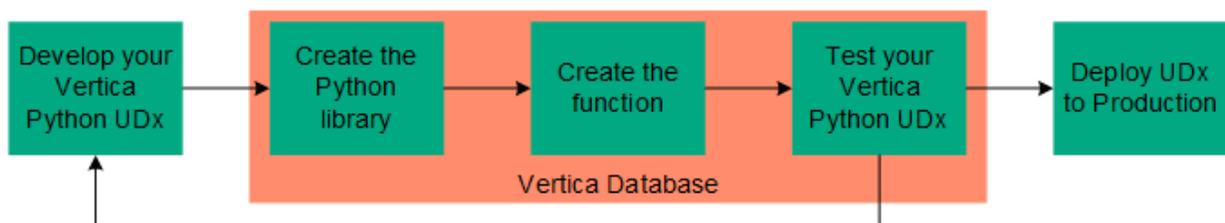
Notes

- The `jvm` resource pool is used only to allocate memory for the Java UDX function calls in a statement. The rest of the resources required by the SQL statement come from other memory pools.
- The first time a Java UDX is called, Vertica starts a JVM to execute some Java methods to get metadata about the UDX during the query planning phase. The memory for this JVM is also taken from the `jvm` memory pool.

Developing with the Python SDK

The Vertica Python SDK provides new capabilities to the Vertica Analytics Platform. Unlike the other Vertica Software Development Kits (SDKs), the Vertica Python SDK does not require any additional system configuration or header files. This low overhead allows you to develop and deploy new capabilities to your Vertica cluster in a short amount of time.

The following workflow is typical for the Python SDK:



Because Python is an interpreted language, you do not have to compile your program before loading the UDX in Vertica. However, you should expect to do some debugging of your code after you create your function and begin testing it in Vertica.

You can find detailed documentation of all of the classes in the Vertica Python SDK [Documentation](#).

Python Libraries

Before you can use your Python UDx, you need to verify that it meets the following library requirements:

- Your UDx must import the `vertica_sdk` package in your code. You do not need to download this package. It is included as a part of the Vertica server.

```
import vertica_sdk
```

- The Vertica Python SDK includes the Python Standard Library. Be aware that importing a package other than the Python Standard Library results in failure.

Python and Vertica Data Types

The Vertica Python SDK converts native Vertica data types into the appropriate Python data types, as follows:

Vertica Data Type	Python Data Type
INTEGER	int
FLOAT	float
NUMERIC	decimal.Decimal
DATE	datetime.date
CHAR, VARCHAR, LONG VARCHAR	string (UTF-8 encoded)
BINARY, VARBINARY, LONG VARBINARY	binary
TIMESTAMP	datetime.datetime
TIME	datetime.time

Note: Some Vertica Analytics Platform data types are not supported in Python.

Handling Errors

If your UDX encounters an unrecoverable error, it should throw a `UdfException`. The exception triggers a rollback of the function call. In Python, this function rollback occurs when the UDX raises an exception.

The following code shows how you can add error checking to your UDX. In this example, if one of the arguments is less than 100, then the Python UDX throws an error.

```
while(True):
    # Example of error checking best practices.
    product_id = block_reader.getInt(2)
    if product_id < 100:
        raise ValueError("Invalid Product ID")
```

When an exception is raised in your Python UDX, the UDX throws a `UdfException` and generates an error message.

```
=> SELECT add2ints(prod_cost, sale_price, product_id) FROM bunch_of_numbers;
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error calling processBlock() in User
Defined Object [add2ints]
at [/scratch_a/release/svrtar11244/vbuild/vertica/OSS/UDxFence/PythonInterface.cpp:168], error code:
0,
message: Error [/scratch_a/release/svrtar11244/vbuild/vertica/OSS/UDxFence/PythonInterface.cpp:385]
function ['call_method']
(Python error type [<class 'ValueError'>])
Traceback (most recent call last):
  File "/home/dbadmin/py_db/v_py_db_node0001_
catalog/Libraries/02fc4af0ace6f91eefa74baecf3ef76000a000000004fc4/pylib_
02fc4af0ace6f91eefa74baecf3ef76000a000000004fc4.py",
line 13, in processBlock
    raise ValueError("Invalid Product ID")
ValueError: Invalid Product ID
```

See Also

[Writing Messages to Log Files](#)

Writing Messages to Log Files

Writing messages to a log can help you when you debug your Python UDXs and want to output additional information about an error condition.

To write a message to the vertica log file, use the `server_interface.log()` function:

```
def processBlock(self, server_interface, arg_reader, res_writer):
    server_interface.log("Python UDX - Adding 2 ints!")
    while(True):
        first_int = block_reader.getInt(0)
        second_int = block_reader.getInt(1)
        block_writer.setInt(first_int + second_int)
        server_interface.log("Values: first_int is {} second_int is {}".format(first_int, second_int))
        block_writer.next()
        if not block_reader.next():
            break
```

Verticawrites the messages to a log file stored in the catalog directory's `UDxLog` subdirectory, which is named `UDxFencedProcesses.log`:

```
$ tail /home/dbadmin/py_db/v_py_db_node0001_catalog/UDxLogs/UDxFencedProcesses.log
07:52:12.862 [Python-v_py_db_node0001-7524:0x206c-40575] 0x7f70eee2f780
PythonExecContext::processBlock
07:52:12.862 [Python-v_py_db_node0001-7524:0x206c-40575] 0x7f70eee2f780 [UserMessage] add2ints -
Python UDX - Adding 2 ints!
07:52:12.862 [Python-v_py_db_node0001-7524:0x206c-40575] 0x7f70eee2f780 [UserMessage] add2ints -
Values: first_int is 100 second_int is 100
```

Vertica adds the SQL name of the UDX to the log message. It also adds the string `[UserMessage]` to mark the entry as a message added by a call to the `server_interface.log()` function. These additions allow you to filter the log to find the messages generated by your UDX.

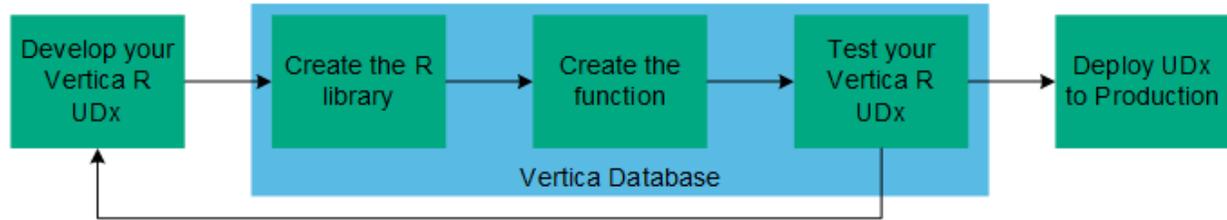
See Also

[Handling Errors](#)

Developing with the R SDK

The Vertica R SDK extends the capabilities of the Vertica Analytics Platform so you can leverage additional R libraries. Before you can begin developing User Defined Extensions (UDxs) in R, you must install the R Language Pack for Vertica on each of the nodes in your cluster. The R SDK supports scalar and transform functions in fenced mode. Other UDX types are not supported.

The following workflow is typical for the R SDK:



You can find detailed documentation of all of the classes in the [Vertica R SDK API Documentation](#).

Installing/Upgrading the R Language Pack for Vertica

To create R UDXs in Vertica, install the R Language Pack package that matches your server version. The R Language Pack includes the R runtime and associated libraries for interfacing with Vertica.

You must install the R Language Pack on each node in the cluster. The Vertica R Language Pack must be the only R Language Pack installed on the node.

Vertica R Language Pack Prerequisites

The R Language Pack RPM requires `libgfortran.so.1`, which may not be installed by default on your system. Install the RPM that contains `libgfortran.so.1`.

See the table below to determine how to install `libgfortran.so.1`.

Linux Version	How to Install libgfortran
RHEL 6/7 and CentOS 6/7	Install the <code>compat-libgfortran-41</code> RPM with the command: <pre>\$ yum install compat-libgfortran-41</pre>
Other supported platforms that use yum, such as SUSE.	You can determine the RPM needed for <code>libgfortran.so.1</code> with the command: <pre>\$ yum whatprovides /usr/lib64/libgfortran.so.1</pre> <p>Typical packages that include <code>libgfortran.so.1</code> include:</p> <ul style="list-style-type: none"><code>libgfortran-41-<any_minor_version>.rpm</code><code>compat-libgfortran-41-<any_minor_</code>

Linux Version	How to Install libgfortran
	<pre>version>.rpm;</pre> <ul style="list-style-type: none">• gcc41-fortran.rpm

Installing the Vertica R Language Pack

1. Download the R language package by going to the [myVertica portal](#), clicking the downloads tab, and selecting the `vertica-R-lang_version.rpm` (or `.deb`) file for your server version. The R language package version must match your server version to three decimal points.
2. Install the package as root or using sudo:

- RHEL/CentOS

```
$ rpm -Uvh vertica-R-lang_version.rpm
```

- Debian

```
$ dpkg -i vertica-R-lang_version.deb
```

The installer puts the R binary in `/opt/vertica/R`. The installer also adds the file `vertica-udx-R.conf` to `/etc/ld.so.conf.d/`. This file is removed if you uninstall the package.

Upgrading the Vertica R Language Pack

When upgrading, some R packages you have manually installed may not work and may have to be reinstalled. If you do not update your package(s), then R returns an error if the package cannot be used. Instructions for upgrading these packages are below.

Note: The R packages provided in the R Language Pack are automatically upgraded and do not need to be reinstalled.

1. You must uninstall the R Language package before upgrading Vertica. Any additional R packages you manually installed remain in `/opt/vertica/R` and are not removed when you uninstall the package.
2. Upgrade your server package as detailed in [Upgrading Vertica to a New Version](#).

3. After the server package has been updated, install the new R Language package on each host.

If you have installed additional R packages, on each node:

1. As root run `/opt/vertica/R/bin/R` and issue the command:

```
> update.packages(checkBuilt=TRUE)
```

2. Select a CRAN mirror from the list displayed.
3. You are prompted to update each package that has an update available for it. You must update any packages that you manually installed and are not compatible with the current version of R in the R Language Pack.

Do **NOT** update:

- Rcpp
- Rinside

The packages you selected to be updated are installed. Quit R with the command:

```
> quit()
```

Vertica UDX functions written in R do not need to be compiled and you do not need to reload your Vertica-R libraries and functions after an upgrade.

R Packages

The Vertica R Language Pack includes the following R packages in addition to the default packages bundled with R:

- Rcpp
- Rinside
- IpSolve
- IpSolveAPI

You can install additional R packages not included in the Vertica R Language Pack by using one of two methods. You must install the same packages on all nodes.

Installing R Packages

You can install additional R packages by using one of the two following methods.

Using the `install.packages()` R command:

```
$ sudo /opt/vertica/R/bin/R  
> install.packages("Zelig");
```

Using CMD INSTALL:

```
/opt/vertica/R/bin/R CMD INSTALL <path-to-package-tgz>
```

The installed packages are located in: `/opt/vertica/R/library`.

R and Vertica Data Types

The following data types are supported when passing data to/from an R UDX:

Vertica Data Type	R Data Type
BOOLEAN	logical
DATE, DATETIME, SMALLDATETIME, TIME, TIMESTAMP, TIMESTAMPZ, TIMETZ	numeric
DOUBLE PRECISION, FLOAT, REAL	numeric
BIGINT, DECIMAL, INT, NUMERIC, NUMBER, MONEY	numeric
BINARY, VARBINARY	character
CHAR, VARCHAR	character

NULL values in Vertica are translated to R NA values when sent to the R function. R NA values are translated into Vertica null values when returned from the R function to Vertica.

Important: When specifying LONG VARCHAR or LONG VARBINARY data types, include the space between the two words. For example, `datatype = c("long varchar")`.

Handling Errors

If your UDX encounters an unrecoverable error, it should throw a `UdfException`. The error triggers a rollback of the function call. In R, this function rollback occurs when the UDX executes an error action.

The following code shows how you can add error checking to your UDX. In this example, if the third column of the data frame does not match the specified Product ID, then the R UDX throws an error.

```
Calculate_Cost_w_Tax <- function(input.data.frame) {  
  # Must match the Product ID 11444  
  if ( !is.numeric(input.data.frame[, 3]) == 11444 ) {  
    stop("Invalid Product ID!")  
  } else {  
    cost_w_tax <- data.frame(input.data.frame[, 1] * input.data.frame[, 2])  
  }  
  return(cost_w_tax)  
}  
  
Calculate_Cost_w_TaxFactory <- function() {  
  list(name=Calculate_Cost_w_Tax,  
        udxtype=c("scalar"),  
        intype=c("float","float", "float"),  
        outtype=c("float"))  
}
```

When an exception is raised in your R UDX, the UDX throws a `UdfException` and generates an error message.

```
=> SELECT Calculate_Cost_w_Tax(item_price, tax_rate, prod_id) FROM Inventory_Sales_Data;  
vsq1:sql_test_multiply.sql:21: ERROR 3399: Failure in UDX RPC call InvokeProcessBlock():  
Error calling processBlock() in User Defined Object [mul] at  
[/scratch_a/release/svrtar30318/vbuild/vertica/OSS/UDxFence/RInterface.cpp:1308],  
error code: 0, message: Exception in processBlockForR :Invalid Product ID!
```

Adding Metadata to R Libraries

You can add metadata, such as author name, the version of the library, a description of your library, and so on to your library. This metadata lets you track the version of your function that is deployed on an Vertica Analytics Platform cluster and lets third-party users of your function know who created the function. Your library's metadata appears in the [USER_LIBRARIES](#) system table after your library has been loaded into the Vertica Analytics Platform catalog.

You declare the metadata for your library by calling the `RegisterLibrary()` function in one of the source files for your UDX. If there is more than one function call in the source files for

your UDX, whichever gets interpreted last as Vertica Analytics Platform loads the library is used to determine the library's metadata.

The `RegisterLibrary()` function takes eight string parameters:

```
RegisterLibrary(author,  
                library_build_tag,  
                library_version,  
                library_sdk_version,  
                source_url,  
                description,  
                licenses_required,  
                signature);
```

- `author` contains whatever name you want associated with the creation of the library (your own name or your company's name for example).
- `library_build_tag` is a string you want to use to represent the specific build of the library (for example, the SVN revision number or a timestamp of when the library was compiled). This is useful for tracking instances of your library as you are developing them.
- `library_version` is the version of your library. You can use whatever numbering or naming scheme you want.
- `library_sdk_version` is the version of the Vertica Analytics Platform SDK Library for which you've compiled the library.

Note: This field isn't used to determine whether a library is compatible with a version of the Vertica Analytics Platform server. The version of the Vertica Analytics Platform SDK you use to compile your library is embedded in the library when you compile it. It is this information that Vertica Analytics Platform server uses to determine if your library is compatible with it.

- `source_url` is a URL where users of your function can find more information about it. This can be your company's website, the GitHub page hosting your library's source code, or whatever site you like.
- `description` is a concise description of your library.
- `licenses_required` is a placeholder for licensing information. You must pass an empty string for this value.
- `signature` is a placeholder for a signature that will authenticate your library. You must pass an empty string for this value.

The following example shows how to add metadata to an R UDX.

```
RegisterLibrary("Speedy Analytics Ltd.",  
  "1234",  
  "1.0",  
  "8.1.0",  
  "http://www.example.com/sales_tax_calculator.R",  
  "Sales Tax R Library",  
  "",  
  "",  
  "")
```

Loading the library and querying the `USER_LIBRARIES` system table shows the metadata supplied in the call to `RegisterLibrary`:

```
=> CREATE LIBRARY rLib AS '/home/dbadmin/sales_tax_calculator.R' LANGUAGE 'R';  
CREATE LIBRARY  
=> SELECT * FROM USER_LIBRARIES WHERE lib_name = 'rLib';  
-[ RECORD 1 ]-----+-----  
schema_name      | public  
lib_name         | rLib  
lib_oid         | 45035996273708350  
author          | Speedy Analytics Ltd.  
owner_id        | 45035996273704962  
lib_file_name    | rLib_02552872a35d9352b4907d3fcd03cf9700a00000000d3e.R  
md5_sum         | 30da555537c4d93c352775e4f31332d2  
sdk_version      |  
revision        |  
lib_build_tag    | 1234  
lib_version      | 1.0  
lib_sdk_version  | 8.1.0  
source_url       | http://www.example.com/sales_tax_calculator.R  
description      | Sales Tax R Library  
licenses_required |  
signature        |  
dependencies     |  
is_valid         | t  
sal_storage_id   | 02552872a35d9352b4907d3fcd03cf9700a00000000d3e
```

Setting Null Input and Volatility Behavior for R Functions

Vertica supports defining volatility and null-input settings for UDxs written in R. Both settings aid in the performance of your R function.

Volatility Settings

Volatility settings describe the behavior of the function to the Vertica optimizer. For example, if you have identical rows of input data and you know the UDx is immutable, then you can define the UDx as `IMMUTABLE`. This tells the Vertica optimizer that it can return a cached value for subsequent identical rows on which the function is called rather than having the function run on each identical row.

To indicate your UDX's volatility, set the volatility parameter of your R factory function to one of the following values:

Value	Description
VOLATILE	Repeated calls to the function with the same arguments always result in different values. Vertica always calls volatile functions for each invocation.
IMMUTABLE	Calls to the function with the same arguments always results in the same return value.
STABLE	Repeated calls to the function with the same arguments <i>within the same statement</i> returns the same output. For example, a function that returns the current user name is stable because the user cannot change within a statement. The user name could change between statements.
DEFAULT_VOLATILITY	The default volatility. This is the same as VOLATILE.

If you do not define a volatility, then the function is considered to be VOLATILE.

The following example sets the volatility to STABLE in the multiplyTwoIntsFactory function:

```
multiplyTwoIntsFactory <- function() {
  list(name      = multiplyTwoInts,
       udxtype   = c("scalar"),
       intype    = c("float","float"),
       outtype   = c("float"),
       volatility = c("stable"),
       parametercallback = multiplyTwoIntsParameters)
}
```

Null Input Behavior

Null input setting determine how to respond to rows that have null input. For example, you can choose to return null if any inputs are null rather than calling the function and having the function deal with a NULL input.

To indicate how your UDX reacts to NULL input, set the strictness parameter of your R factory function to one of the following values:

Value	Description
CALLED_ON_NULL_INPUT	The function must be called, even if one or more arguments are NULL.

Value	Description
RETURN_NULL_ON_NULL_INPUT	The function always returns a NULL value if any of its arguments are NULL.
STRICT	A synonym for RETURN_NULL_ON_NULL_INPUT
DEFAULT_STRICTNESS	The default strictness setting. This is the same as CALLED_ON_NULL_INPUT.

If you do not define a null input behavior, then the function is called on every row of data regardless of the presence of NULL values.

The following example sets the NULL input behavior to STRICT in the multiplyTwoIntsFactory function:

```
multiplyTwoIntsFactory <- function() {  
  list(name      = multiplyTwoInts,  
       udxtype   = c("scalar"),  
       intype    = c("float","float"),  
       outtype   = c("float"),  
       strictness = c("strict"),  
       parametertypecallback = multiplyTwoIntsParameters)  
}
```

Aggregate Functions (UDAFs)

Aggregate functions perform an operation on a set of values and return one value. Vertica provides standard built-in aggregate functions such as [AVG](#), [MAX](#), and [MIN](#). User-Defined Aggregate Functions work similarly to the built-in aggregate functions.

User-Defined Aggregate Functions:

- Support a single input column (or set) of values and provide a single output column.
- Support RLE decompression; RLE input is decompressed before it is sent to a UDAF.
- Can be used with the GROUP BY and HAVING clauses. Only columns appearing in the GROUP BY clause can be selected.
- Cannot be used with correlated subquery.

UDAFs are available for C++ only.

UDAF Class Overview

You create your UDAF by subclassing two classes defined by the Vertica SDK: `AggregateFunction` and `AggregateFunctionFactory`.

AggregateFunction

The `AggregateFunction` class performs the aggregation. It computes values on each database node where relevant data is stored and then combines the results from the nodes. You must implement the following methods:

- `initAggregate()` - Initializes the class, defines variables, and sets the starting value for the variables. This function must be idempotent.
- `aggregate()` - The main aggregation operation, executed on each node.
- `combine()` - If multiple invocations of `aggregate()` are needed, Vertica calls `combine()` to combine all the sub-aggregations into a final aggregation. Although this method might

not be called, you must define it.

- `terminate()` - Terminates the function and returns the result as a column.

Important: The `aggregate()` function might not operate on the complete input set all at once. For this reason, `initAggregate()` must be idempotent.

The `AggregateFunction` class also provides optional methods that you can implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDAF API (see [Allocating Resources for UDFs](#) for details).

AggregateFunctionFactory

The `AggregateFunctionFactory` class specifies metadata information such as the argument and return types of your aggregate function. It also instantiates your `AggregateFunction` subclass. Your subclass must implement the following methods:

- `getPrototype()` - Defines the number of parameters and data types accepted by the function. There is a single parameter for aggregate functions.
- `getIntermediateTypes()` - Defines the intermediate variable(s) used by the function. These variables are used when combining the results of `aggregate()` calls.
- `getParameterType()` - Defines the names and types of parameters that this function uses (optional).
- `getReturnType()` - Defines the type of the output column.

Vertica uses this data when you call the [CREATE AGGREGATE FUNCTION](#) SQL statement to add the function to the database catalog.

UDAF Performance in Statements Containing a GROUP BY Clause

You may see slower-than-expected performance from your UDAF if the SQL statement calling it also contains a [GROUP BY Clause](#). For example:

```
=> SELECT a, MYUDAF(b) FROM sampletable GROUP BY a;
```

In statements like this one, Vertica does not consolidate row data together before calling your UDAF's `aggregate()` method. Instead, it calls `aggregate()` once for each row of data. Usually, the overhead of having Vertica consolidate the row data is greater than the overhead of calling `aggregate()` for each row of data. However, if your UDAF's `aggregate()` method has significant overhead, then you might notice an impact on your UDAF's performance.

For example, suppose `aggregate()` allocates memory. When called in a statement with a `GROUP BY` clause, it performs this memory allocation for each row of data. Because memory allocation is a relatively expensive process, this allocation can impact the overall performance of your UDAF and the query.

There are two ways you can address UDAF performance in a statement containing a `GROUP BY` clause:

- Reduce the overhead of each call to `aggregate()`. If possible, move any allocation or other setup operations to the UDAF's `setup()` function.
- Declare a special parameter that tells Vertica to group row data together when calling a UDAF. This technique is explained below.

Using the `_minimizeCallCount` Parameter

Your UDAF can tell Vertica to always batch row data together to reduce the number of calls to its `aggregate()` method. To trigger this behavior, your UDAF must declare an integer parameter named `_minimizeCallCount`. You do not need to set a value for this parameter in your SQL statement. The fact that your UDAF declares this parameter triggers Vertica to group row data together when calling `aggregate()`.

You declare the `_minimizeCallCount` parameter the same way you declare other UDF parameters. See [UDF Parameters](#) for more information.

Important: Always test the performance of your UDAF before and after implementing the `_minimizeCallCount` parameter to ensure that it improves performance. You might find that the overhead of having Vertica group row data for your UDAF is greater than the cost of the repeated calls to `aggregate()`.

C++ API

This section provides APIs and examples for the C++ API for UDAFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

AggregateFunction and AggregateFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See [UDAF Class Overview](#) for general information about implementing the `AggregateFunction` and `AggregateFunctionFactory` classes.

AggregateFunction API

The API provides the following methods for extension by subclasses:

```
virtual void setup(ServerInterface &srvInterface,
                  const SizedColumnTypes &argTypes);

virtual void initAggregate(ServerInterface &srvInterface, IntermediateAggs &aggs)=0;

void aggregate(ServerInterface &srvInterface, BlockReader &arg_reader,
              IntermediateAggs &aggs);

virtual void combine(ServerInterface &srvInterface, IntermediateAggs &aggs_output,
                   MultipleIntermediateAggs &aggs_other)=0;

virtual void terminate(ServerInterface &srvInterface, BlockWriter &res_writer,
                     IntermediateAggs &aggs);

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes);
```

AggregateFunctionFactory API

The API provides the following methods for extension by subclasses:

```
virtual AggregateFunction *
    createAggregateFunction(ServerInterface &srvInterface)=0;

virtual void getPrototype(ServerInterface &srvInterface,
                        ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getIntermediateTypes(ServerInterface &srvInterface,
                                const SizedColumnTypes &inputTypes, SizedColumnTypes &intermediateTypeMetaData)=0;

virtual void getReturnType(ServerInterface &srvInterface,
                          const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0;

virtual void getParameterType(ServerInterface &srvInterface,
                             SizedColumnTypes &parameterTypes);
```

C++ Example: Average

The Average aggregate function created in this example computes the average of values in a column.

You can find the source code used in this example on the [Vertica GitHub page](#).

Loading the Example

Use `CREATE LIBRARY` and `CREATE AGGREGATE FUNCTION` to declare the function:

```
=> CREATE LIBRARY AggregateFunctions AS
'/opt/vertica/sdk/examples/build/AggregateFunctions.so';
CREATE LIBRARY
=> CREATE aggregate function ag_avg AS LANGUAGE 'C++'
name 'AverageFactory' library AggregateFunctions;
CREATE AGGREGATE FUNCTION
```

Using the Example

Use the function as part of a `SELECT` statement:

```
=> SELECT * FROM average;
id | count
----+-----
A |      8
B |      3
C |      6
D |      2
E |      9
F |      7
G |      5
H |      4
I |      1
(9 rows)
=> SELECT ag_avg(count) FROM average;
ag_avg
-----
5
(1 row)
```

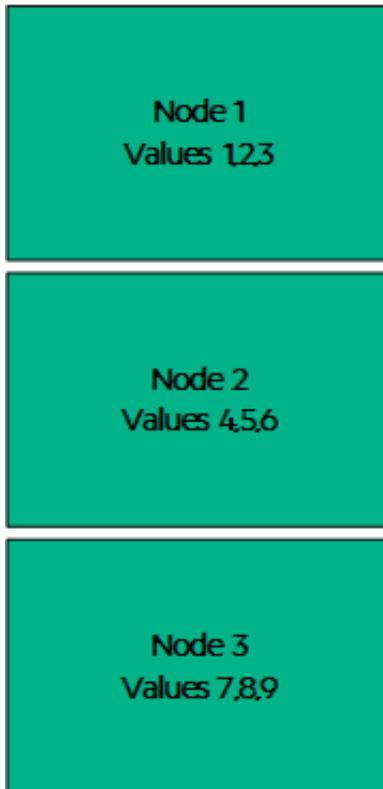
AggregateFunction Implementation

This example adds the input argument values in the `aggregate()` method and keeps a counter of the number of values added. The server runs `aggregate()` on every node and different data chunks, and combines all the individually added values and counters in the

`combine()` method. Finally, the average value is computed in the `terminate()` method by dividing the total sum by the total number of values processed.

For this discussion, assume the following environment:

- A three-node Vertica cluster
- A table column that contains nine values that are evenly distributed across the nodes. Schematically, the nodes look like the following figure:



The function uses `sum` and `count` variables. `sum` contains the sum of the values, and `count` contains the count of values.

First, `initAggregate()` initializes the variables and sets their values to zero.

```
virtual void initAggregate(ServerInterface &srvInterface,
                          IntermediateAggs &aggs)
{
  try {
    VNumeric &sum = aggs.getNumericRef(0);
    sum.setZero();

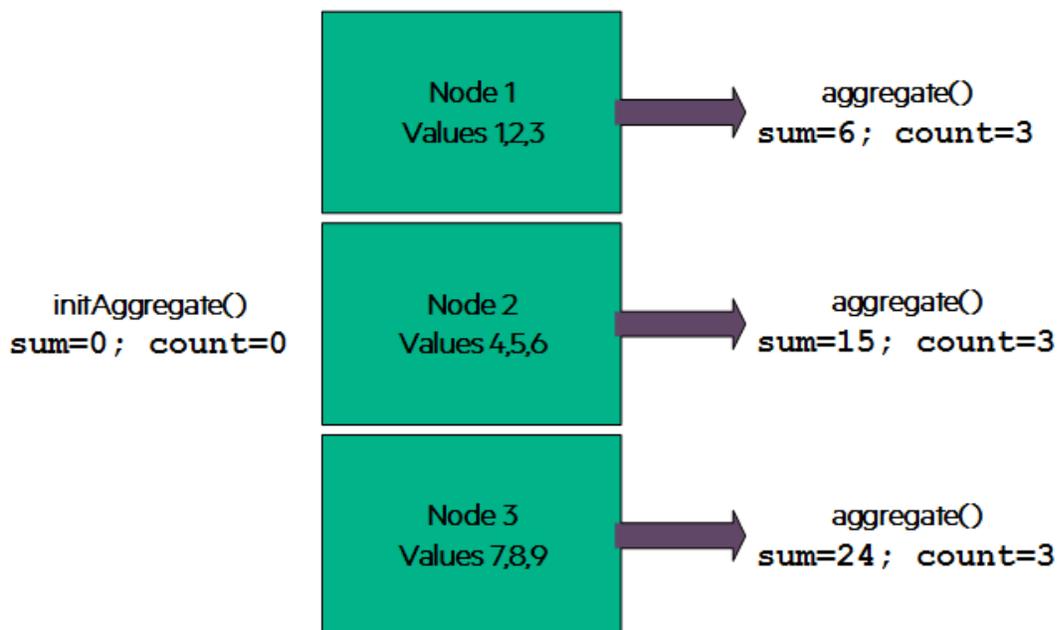
    vint &count = aggs.getIntRef(1);
    count = 0;
  }
  catch(std::exception &e) {
    vt_report_error(0, "Exception while initializing intermediate aggregates: [% s]", e.what());
  }
}
```

```
}  
}
```

The `aggregate()` function reads the block of data on each node and calculates partial aggregates.

```
void aggregate(ServerInterface &srvInterface,  
              BlockReader &argReader,  
              IntermediateAggs &aggs)  
{  
  try {  
    VNumeric &sum = aggs.getNumericRef(0);  
    vint &count = aggs.getIntRef(1);  
  
    do {  
      const VNumeric &input = argReader.getNumericRef(0);  
      if (!input.isNull()) {  
        sum.accumulate(&input);  
        count++;  
      }  
    } while (argReader.next());  
  } catch (std::exception &e) {  
    vt_report_error(0, "Exception while processing aggregate: [%s]", e.what());  
  }  
}
```

Each completed instance of the `aggregate()` function returns multiple partial aggregates for sum and count. The following figure illustrates this process using the `aggregate()` function:



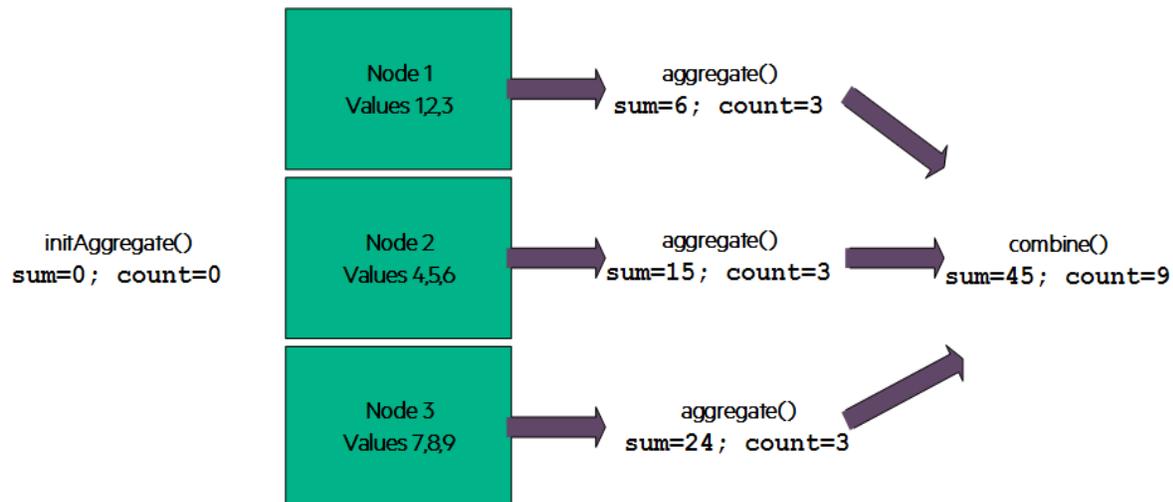
The `combine()` function puts together the partial aggregates calculated by each instance of the average function.

```

virtual void combine(ServerInterface &srvInterface,
                   IntermediateAggs &aggs,
                   MultipleIntermediateAggs &aggsOther)
{
  try {
    VNumeric &mySum = aggs.getNumericRef(0);
    vint &myCount = aggs.getIntRef(1);
    do {
      const VNumeric &otherSum = aggsOther.getNumericRef(0);
      const vint &otherCount = aggsOther.getIntRef(1);
      mySum.accumulate(&otherSum);
      myCount += otherCount;
    } while (aggsOther.next());
  } catch (std::exception &e) {
    vt_report_error(0, "Exception while combining intermediate
    aggregates: [%s]", e.what());
  }
}

```

The following figure shows how each partial aggregate is combined:



After all input has been evaluated by the `aggregate()` function Vertica calls the `terminate()` function. It returns the average to the caller.

```

virtual void terminate(ServerInterface &srvInterface,
                     BlockWriter &resWriter,
                     IntermediateAggs &aggs)
{
  try {
    const int32 MAX_INT_PRECISION = 20;
    const int32 prec = Basics::getNumericWordCount(MAX_INT_PRECISION);
    uint64 words[prec];
    VNumeric count(words, prec, 0/*scale*/);
    count.copy(aggs.getIntRef(1));

    VNumeric &out = resWriter.getNumericRef();
    if (count.isZero()) {

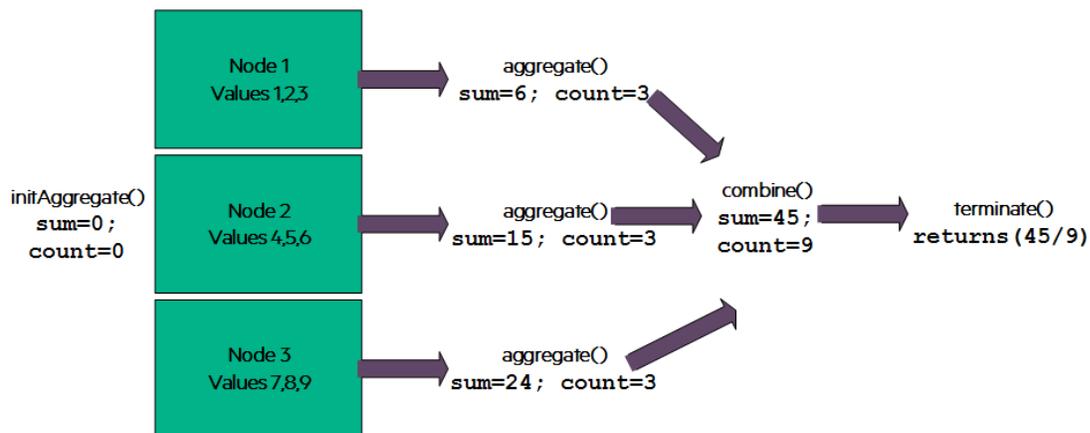
```

```

    out.setNull();
  } else
    const VNumeric &sum = aggs.getNumericRef(0);
    out.div(&sum, &count);
  }
}

```

The following figure shows the implementation of the `terminate()` function:



AggregateFunctionFactory Implementation

The `getPrototype()` function allows you to define the variables that are sent to your aggregate function and returned to Vertica after your aggregate function runs. The following example accepts and returns a numeric value:

```

virtual void getPrototype(ServerInterface &srvfloaterface,
    ColumnTypes &argTypes,
    ColumnTypes &returnType)
{
    argTypes.addNumeric();
    returnType.addNumeric();
}

```

The `getIntermediateTypes()` function defines any intermediate variables that you use in your aggregate function. *Intermediate variables* are values used to pass data among multiple invocations of an aggregate function. They are used to combine results until a final result can be computed. In this example, there are two results - total (numeric) and count (int).

```

virtual void getIntermediateTypes(ServerInterface &srvInterface,
    const SizedColumnTypes &inputTypes,
    SizedColumnTypes &intermediateTypeMetaData)
{
    const VerticaType &inType = inputTypes.getColumnType(0);
    intermediateTypeMetaData.addNumeric(interPrec, inType.getNumericScale());
    intermediateTypeMetaData.addInt();
}

```

```
}
```

The `getReturnType()` function defines the output data type:

```
virtual void getReturnType(ServerInterface &srvfloaterface,  
                           const SizedColumnTypes &inputTypes,  
                           SizedColumnTypes &outputTypes)  
{  
    const VerticaType &inType = inputTypes.getColumnType(0);  
    outputTypes.addNumeric(inType.getNumericPrecision(),  
                           inType.getNumericScale());  
}
```

Analytic Functions (UDAnFs)

User-Defined Analytic Functions (UDAnFs) are used for analytics. See [SQL Analytics](#) for an overview of Vertica's built-in analytics. Like User-Defined Scalar Functions (UDSFs), UDAnFs must output a single value for each row of data read and can have no more than 1600 arguments.

Unlike UDSFs, the UDAnF's input reader and output reader can be advanced independently. This feature lets you create UDAnF's where the output value is calculated over multiple rows of data. By advancing the reader and writer independently, you can create functions similar to the built-in analytic functions such as [LAG](#), which uses data from prior rows to output a value for the current row.

UDAnF Class Overview

You create your UDAnF by subclassing two classes defined by the Vertica SDK: `AnalyticFunction` and `AnalyticFunctionFactory`.

AnalyticFunction

The `AnalyticFunction` subclass performs the analytic processing. Your subclass must define the `processPartition()` method to perform the operation. It may define methods to set up and tear down the function.

Performing the Operation

The `processPartition()` method reads a partition of data, performs some sort of processing, and outputs a single value for each input row.

Vertica calls `processPartition()` once for each partition of data. It supplies the partition using an `AnalyticPartitionReader` object from which you read its input data. In addition, there is a unique method on this object named `isNewOrderByKey()`, which returns a Boolean value indicating whether your function has seen a row with the same ORDER BY key (or keys). This method is very useful for analytic functions (such as the example RANK function) which need to handle rows with identical ORDER BY keys differently than rows with different ORDER BY keys.

Note: You can specify multiple ORDER BY columns in the SQL query you use to call your UDA nF. The `isNewOrderByKey` method returns true if any of the ORDER BY keys are different than the previous row.

Once your method has finished processing the row of data, you advance it to the next row of input by calling `next()` on `AnalyticPartitionReader`.

Your method writes its output value using an `AnalyticPartitionWriter` object that Vertica supplies as a parameter to `processPartition()`. This object has data-type-specific methods to write the output value (such as `setInt()`). After setting the output value, call `next()` on `AnalyticPartitionWriter` to advance to the next row in the output.

Note: You must be sure that your function produces a row of output for each row of input in the partition. You must also not output more rows than are in the partition, otherwise the zygote size process (if running in [Fenced Mode](#)) or Vertica itself could generate an out of bounds error.

Setting Up and Tearing Down

The `AnalyticFunction` class defines two additional methods that you can optionally implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDx API (see [Allocating Resources for UDxs](#) for details).

AnalyticFunctionFactory

The `AnalyticFunctionFactory` class tells Vertica metadata about your UDA nF: its number of parameters and their data types, as well as the data type of its return value. It also

instantiates a subclass of `AnalyticFunction`.

Your `AnalyticFunctionFactory` subclass must implement the following methods:

- `getPrototype()` describes the input parameters and output value of your function. You set these values by calling functions on two `ColumnTypes` objects that are passed to your method.
- `createAnalyticFunction()` supplies an instance of your `AnalyticFunction` that Vertica can call to process a UDAF function call.
- `getReturnType()` provides details about your function's output. This method is where you set the width of the output value if your function returns a variable-width value (such as `VARCHAR`) or the precision of the output value if it has a settable precision (such as `TIMESTAMP`).

C++ API

This section provides APIs and examples for the C++ API for UDAnFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

AnalyticFunction and AnalyticFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See [UDAnF Class Overview](#) for general information about implementing the `AnalyticFunction` and `AnalyticFunctionFactory` classes.

AnalyticFunction API

The API provides the following methods for extension by subclasses:

```
virtual void setup(ServerInterface &srvInterface,  
                  const SizedColumnTypes &argTypes);  
  
virtual void processPartition (ServerInterface &srvInterface,  
                               AnalyticPartitionReader &input_reader,  
                               AnalyticPartitionWriter &output_writer)=0;  
  
virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes);
```

AnalyticFunctionFactory API

The API provides the following methods for extension by subclasses:

```
virtual AnalyticFunction * createAnalyticFunction (ServerInterface &srvInterface)=0;

virtual void getPrototype(ServerInterface &srvInterface,
    ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getReturnType(ServerInterface &srvInterface,
    const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0;

virtual void getParameterType(ServerInterface &srvInterface,
    SizedColumnTypes &parameterTypes);
```

C++ Example: Rank

The Rank analytic function ranks rows based on how they are ordered.

Loading and Using the Example

The following example shows how to load the function into Vertica. It assumes that the `AnalyticFunctions.so` library that contains the function has been copied to the `dbadmin` user's home directory on the initiator node.

```
=> CREATE LIBRARY AnalyticFunctions AS '/home/dbadmin/AnalyticFunctions.so';
CREATE LIBRARY
=> CREATE ANALYTIC FUNCTION an_rank AS LANGUAGE 'C++'
    NAME 'RankFactory' LIBRARY AnalyticFunctions;
CREATE ANALYTIC FUNCTION
```

An example of running this rank function, named `an_rank`, is:

```
=> SELECT * FROM hits;
  site      | date      | num_hits
-----+-----+-----
www.example.com | 2012-01-02 |      97
www.vertica.com | 2012-01-01 |  343435
www.example.com | 2012-01-01 |     123
www.example.com | 2012-01-04 |     112
www.vertica.com | 2012-01-02 |  503695
www.vertica.com | 2012-01-03 |  490387
www.example.com | 2012-01-03 |     123
(7 rows)
=> SELECT site,date,num_hits,an_rank()
    OVER (PARTITION BY site ORDER BY num_hits DESC)
    AS an_rank FROM hits;
  site      | date      | num_hits | an_rank
```

```
-----+-----+-----+-----  
www.example.com | 2012-01-03 |      123 |      1  
www.example.com | 2012-01-01 |      123 |      1  
www.example.com | 2012-01-04 |      112 |      3  
www.example.com | 2012-01-02 |       97 |      4  
www.vertica.com  | 2012-01-02 |   503695 |      1  
www.vertica.com  | 2012-01-03 |   490387 |      2  
www.vertica.com  | 2012-01-01 |   343435 |      3  
(7 rows)
```

As with the built-in [RANK](#) analytic function, rows that have the same value for the ORDER BY column (num_hits in this example) have the same rank, but the rank continues to increase, so that the next row that has a different ORDER BY key gets a rank value based on the number of rows that preceded it.

AnalyticFunction Implementation

The following code defines an `AnalyticFunction` subclass named `Rank`. It is based on example code distributed in the examples directory of the SDK.

```
/**  
 * User-defined analytic function: Rank - works mostly the same as SQL-99 rank  
 * with the ability to define as many order by columns as desired  
 */  
class Rank : public AnalyticFunction  
{  
    virtual void processPartition(ServerInterface &srvInterface,  
        AnalyticPartitionReader &inputReader,  
        AnalyticPartitionWriter &outputWriter)  
    {  
        // Always use a top-level try-catch block to prevent exceptions from  
        // leaking back to Vertica or the fenced-mode side process.  
        try {  
            rank = 1; // The rank to assign a row  
            rowCount = 0; // Number of rows processed so far  
            do {  
                rowCount++;  
                // Do we have a new order by row?  
                if (inputReader.isNewOrderByKey()) {  
                    // Yes, so set rank to the total number of rows that have been  
                    // processed. Otherwise, the rank remains the same value as  
                    // the previous iteration.  
                    rank = rowCount;  
                }  
                // Write the rank  
                outputWriter.setInt(0, rank);  
                // Move to the next row of the output  
                outputWriter.next();  
            } while (inputReader.next()); // Loop until no more input  
        } catch(exception & e) {  
            // Standard exception. Quit.  
            vt_report_error(0, "Exception while processing partition: %s", e.what());  
        }  
    }  
}
```

```
    }  
    private:  
        vint rank, rowCount;  
};
```

In this example, the `processPartition()` method does not actually read any of the data from the input row; it just advances through the rows. It does not need to read data; it just counts the rows that have been read and determine whether those rows have the same ORDER BY key as the previous row. If the current row is a new ORDER BY key, then the rank is set to the total number of rows that have been processed. If the current row has the same ORDER BY value as the previous row, then the rank remains the same.

Note that the function has a top-level try-catch block. All of your UDF functions should always have one to prevent stray exceptions from being passed back to Vertica (if you run the function unfenced) or the side process.

AnalyticFunctionFactory Implementation

The following code defines the `AnalyticFunctionFactory` that corresponds with the Rank analytic function.

```
class RankFactory : public AnalyticFunctionFactory  
{  
    virtual void getPrototype(ServerInterface &srvInterface,  
                             ColumnTypes &argTypes, ColumnTypes &returnType)  
    {  
        returnType.addInt();  
    }  
    virtual void getReturnType(ServerInterface &srvInterface,  
                              const SizedColumnTypes &inputTypes,  
                              SizedColumnTypes &outputTypes)  
    {  
        outputTypes.addInt();  
    }  
    virtual AnalyticFunction *createAnalyticFunction(ServerInterface  
                                                     &srvInterface)  
    { return vt_createFuncObj(srvInterface.allocator, Rank); }  
};
```

The first method defined by the `RankFactory` subclass, `getPrototype()`, sets the data type of the return value. Because the Rank UDAnF does not read input, it does not define any arguments by calling methods on the `ColumnTypes` object passed in the `argTypes` parameter.

The next method is `getReturnType()`. If your function returns a data type that needs to define a width or precision, your implementation of the `getReturnType()` method calls a method on the `SizedColumnType` object passed in as a parameter to tell Vertica the width or precision. Rank returns a fixed-width data type (an `INTEGER`) so it does not need to set the precision or width of its output; it just calls `addInt()` to report its output data type.

Finally, RankFactory defines the `createAnalyticFunction()` method that returns an instance of the `AnalyticFunction` class that Vertica can call. This code is mostly boilerplate. All you need to do is add the name of your analytic function class in the call to `vt_createFuncObj()`, which takes care of allocating the object for you.

Java API

This section provides APIs and examples for the Java API for UDAnFs.

For information on setting up a Java development environment and compiling and packaging libraries, see [Developing with the Java SDK](#).

AnalyticFunction and AnalyticFunctionFactory Java Interface

This section describes information that is specific to the Java API. See [UDAnF Class Overview](#) for general information about implementing the `AnalyticFunction` and `AnalyticFunctionFactory` classes.

AnalyticFunction API

The API provides the following methods for extension by subclasses:

```
void setup(ServerInterface srvInterface, SizedColumnTypes argTypes);

abstract void processPartition (ServerInterface srvInterface,
    AnalyticPartitionReader input_reader, AnalyticPartitionWriter output_writer)
    throws UdfException, DestroyInvocation;

void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes);
```

AnalyticFunctionFactory API

The API provides the following methods for extension by subclasses:

```
abstract AnalyticFunction createAnalyticFunction (ServerInterface srvInterface);

abstract void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType);

abstract void getReturnType(ServerInterface srvInterface, SizedColumnTypes argTypes,
    SizedColumnTypes returnType) throws UdfException;

void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);
```

Java Example: Rank

The Rank analytic function ranks rows based on how they are ordered.

Loading and Using the Example

The following example shows how to load the function into Vertica. It assumes that the `AnalyticFunctions.jar` library that contains the function has been copied to the `dbadmin` user's home directory on the initiator node.

```
=> CREATE LIBRARY AnalyticFunctions AS '/home/dbadmin/AnalyticFunctions.jar';  
CREATE LIBRARY  
=> CREATE ANALYTIC FUNCTION an_rank AS LANGUAGE 'Java'  
    NAME 'RankFactory' LIBRARY AnalyticFunctions;  
CREATE ANALYTIC FUNCTION
```

An example of running this rank function, named `an_rank`, is:

```
=> SELECT * FROM hits;  
   site      | date      | num_hits  
-----+-----+-----  
www.example.com | 2012-01-02 |      97  
www.vertica.com | 2012-01-01 | 343435  
www.example.com | 2012-01-01 |     123  
www.example.com | 2012-01-04 |     112  
www.vertica.com | 2012-01-02 | 503695  
www.vertica.com | 2012-01-03 | 490387  
www.example.com | 2012-01-03 |     123  
(7 rows)  
=> SELECT site,date,num_hits,an_rank()  
    OVER (PARTITION BY site ORDER BY num_hits DESC)  
    AS an_rank FROM hits;  
   site      | date      | num_hits | an_rank  
-----+-----+-----+-----  
www.example.com | 2012-01-03 |     123 |      1  
www.example.com | 2012-01-01 |     123 |      1  
www.example.com | 2012-01-04 |     112 |      3  
www.example.com | 2012-01-02 |      97 |      4  
www.vertica.com | 2012-01-02 | 503695 |      1  
www.vertica.com | 2012-01-03 | 490387 |      2  
www.vertica.com | 2012-01-01 | 343435 |      3  
(7 rows)
```

As with the built-in [RANK](#) analytic function, rows that have the same value for the `ORDER BY` column (`num_hits` in this example) have the same rank, but the rank continues to increase, so that the next row that has a different `ORDER BY` key gets a rank value based on the number of rows that preceded it.

AnalyticFunction Implementation

The following code defines an `AnalyticFunction` subclass named `Rank`. The code can be found in the SDK examples directory.

```
/**
 * User-defined analytic function: Rank - works mostly the same as SQL-99 rank
 * with the ability to define as many order by columns as desired
 *
 */
public class Rank extends AnalyticFunction {
    private int rank, numRowsWithSameOrder;

    @Override
    public void processPartition(ServerInterface srvInterface,
        AnalyticPartitionReader inputReader,
        AnalyticPartitionWriter outputWriter)
        throws UdfException, DestroyInvocation {
        rank = 0;
        numRowsWithSameOrder = 1;
        do{
            if(!inputReader.isNewOrderByKey()){
                ++numRowsWithSameOrder;
            }else {
                rank += numRowsWithSameOrder;
                numRowsWithSameOrder = 1;
            }
            outputWriter.setLong(0, rank);
            outputWriter.next();
        }while(inputReader.next());
    }
}
```

In this example, the `processPartition()` method does not actually read any of the data from the input row; it just advances through the rows. It just needs to count the number of rows that have been read and determine whether those rows have the same `ORDER BY` key as the previous row. If the current row is a new `ORDER BY` key, then the rank is set to the total number of rows that have been processed. If the current row has the same `ORDER BY` value as the previous row, then the rank remains the same.

AnalyticFunctionFactory Implementation

The following code defines the `AnalyticFunctionFactory` that corresponds with the `Rank` analytic function.

```
public class RankFactory extends AnalyticFunctionFactory {

    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes,
```

```
        ColumnTypes returnType) {
    returnType.addInt();
}

@Override
public void getReturnType(ServerInterface srvInterface,
    SizedColumnTypes argTypes,
    SizedColumnTypes returnType) throws UdfException {
    returnType.addInt();
}

@Override
public AnalyticFunction createAnalyticFunction(ServerInterface srvInterface) {
    return new Rank();
}
}
```

The first method defined by the RankFactory subclass, `getPrototype()`, sets the data type of the return value. Because the Rank UDAF does not read input, it does not define any arguments by calling methods on the `ColumnTypes` object passed in the `argTypes` parameter.

The next method is `getReturnType()`. If your analytic function returns a data type that needs to define a width or precision, your implementation of the `getReturnType` function calls a method on the `SizedColumnType` parameter to tell Vertica the width or precision. Rank returns a fixed-width data type (an `INTEGER`) so it does not need to set the precision or width of its output; it just calls `addInt()` to report its output data type.

Finally, RankFactory defines the `createAnalyticFunction()` method that returns an instance of the `AnalyticFunction` class that Vertica can call.

Scalar Functions (UDSFs)

A User-Defined Scalar Function (UDSF) returns a single value for each row of data it reads. You can use a UDSF anywhere you can use a built-in Vertica function. You usually develop a UDSF to perform data manipulations that are too complex or too slow to perform using SQL statements and functions. UDSFs also let you use analytic functions provided by third-party libraries within Vertica while still maintaining high performance.

Your UDSF must return a value for every input row (unless it generates an error; see [Handling Errors](#) for details). Failing to return a value for a row will result in incorrect results and potentially destabilize the Vertica server if not run in [Fenced Mode](#).

A UDSF cannot have more than 1600 arguments.

UDSF Class Overview

You create your UDSF by subclassing two classes defined by the Vertica SDK: `ScalarFunction` and `ScalarFunctionFactory`.

ScalarFunction

The `ScalarFunction` class is the heart of a UDSF. Your subclass must define the `processBlock()` method to perform the scalar operation. It may define methods to set up and tear down the function.

Performing the Operation

The `processBlock()` method carries out all of the processing that you want your UDSF to perform. When a user calls your function in a SQL statement, Vertica bundles together the data from the function parameters and passes it to `processBlock()`.

The input and output of the `processBlock()` method are supplied by objects of the `BlockReader` and `BlockWriter` classes. They define methods that you use to read the input data and write the output data for your UDSF.

The majority of the work in developing a UDSF is writing `processBlock()`. This is where all of the processing in your function occurs. Your UDSF should follow this basic pattern:

- Read in a set of parameters from the `BlockReader` object using data-type-specific methods.
- Process the data in some manner.
- Output the resulting value using one of the `BlockWriter` class's data-type-specific methods.
- Advance to the next row of output and input by calling `BlockWriter.next()` and `BlockReader.next()`.

This process continues until there are no more rows of data to be read (`BlockReader.next()` returns false).

You must make sure that `processBlock()` reads all of the rows in its input and outputs a single value for each row. Failure to do so can corrupt the data structures that Vertica reads to get the output of your UDSF. The only exception to this rule is if your `processBlock()` function reports an error back to Vertica (see [Handling Errors](#)). In that case, Vertica does not attempt to read the incomplete result set generated by the UDSF.

Setting Up and Tearing Down

The `ScalarFunction` class defines two additional methods that you can optionally implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDx API (see [Allocating Resources for UDxs](#) for details).

Notes

- While the name you choose for your `ScalarFunction` subclass does not have to match the name of the SQL function you will later assign to it, Micro Focus considers making the names the same a best practice.
- Do not assume that your function will be called from the same thread that instantiated it.
- The same instance of your `ScalarFunction` subclass can be called on to process multiple blocks of data.
- The rows of input sent to `processBlock()` are not guaranteed to be any particular order.
- Writing too many output rows can cause Vertica to emit an out-of-bounds error.

ScalarFunctionFactory

The `ScalarFunctionFactory` class tells Vertica metadata about your UDSF: its number of parameters and their data types, as well as the data type of its return value. It also instantiates a subclass of `ScalarFunction`.

Methods

You must implement the following methods in your `ScalarFunctionFactory` subclass:

- `createScalarFunction()` instantiates a `ScalarFunction` subclass. If writing in C++, you can call the `vt_createFuncObj` macro with the name of the `ScalarFunction` subclass. This macro takes care of allocating and instantiating the class for you.
- `getPrototype()` tells Vertica about the parameters and return type for your UDSF. In addition to a `ServerInterface` object, this method gets two `ColumnTypes` objects. All you need to do in this function is to call class functions on these two objects to build the list of parameters and the single return value type.

After defining your factory class, you need to call the `RegisterFactory` macro. This macro instantiates a member of your factory class, so Vertica can interact with it and extract the metadata it contains about your UDSF.

Declaring Return Values

If your function returns a sized column (a return data type whose length can vary, such as a `VARCHAR`) or a value that requires precision, you must implement `getReturnType()`. This method is called by Vertica to find the length or precision of the data being returned in each row of the results. The return value of this method depends on the data type your `processBlock()` method returns:

- `CHAR` or `VARCHAR` return the maximum length of the string.
- `NUMERIC` types specify the precision and scale.
- `TIME` and `TIMESTAMP` values (with or without timezone) specify precision.
- `INTERVAL YEAR TO MONTH` specifies range.
- `INTERVAL DAY TO SECOND` specifies precision and range.

If your UDSF does not return one of these data types, it does not need a `getReturnType()` method.

The input to the `getReturnType()` method is a `SizedColumnTypes` object that contains the input argument types along with their lengths. This object will be passed to an instance of your `processBlock()` function. Your implementation of `getReturnType()` must extract the data types and lengths from this input and determine the length or precision of the output rows. It then saves this information in another instance of the `SizedColumnTypes` class.

Setting Null Input and Volatility Behavior

Normally, Vertica calls your UDSF for every row of data in the query. There are some cases where Vertica can avoid executing your UDSF. You can tell Vertica when it can skip calling your function and just supply a return value itself by changing your function's volatility and strictness settings.

- Your function's **volatility** indicates whether it always returns the same output value when passed the same arguments. Depending on its behavior, Vertica can cache the arguments and the return value. If the user calls the UDSF with the same set of arguments, Vertica returns the cached value instead of calling your UDSF.
- Your function's **strictness** indicates how it reacts to NULL arguments. If it always returns NULL when *any* argument is NULL, Vertica can just return NULL without having to call the function. This optimization also saves you work, because you do not need to test for and handle null arguments in your UDSF code.

You indicate the volatility and null handling of your function by setting the `vol` and `strict` fields in your `ScalarFunctionFactory` class's constructor.

Volatility Settings

To indicate your function's volatility, set the `vol` field to one of the following values:

Value	Description
VOLATILE	Repeated calls to the function with the same arguments always result in different values. Vertica always calls volatile functions for each invocation.
IMMUTABLE	Calls to the function with the same arguments always results in the same return value.
STABLE	Repeated calls to the function with the same arguments <i>within the same</i>

Value	Description
	<i>statement</i> returns the same output. For example, a function that returns the current user name is stable because the user cannot change within a statement. The user name could change between statements.
DEFAULT_VOLATILITY	The default volatility. This is the same as VOLATILE.

C++ Example

The following example shows a version of the Add2ints example factory class that makes the function immutable.

```
class Add2intsImmutableFactory : public Vertica::ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
    { return vt_createFuncObj(srvInterface.allocator, Add2ints); }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
        Vertica::ColumnTypes &argTypes,
        Vertica::ColumnTypes &returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

public:
    Add2intsImmutableFactory() {vol = IMMUTABLE;}
};
RegisterFactory(Add2intsImmutableFactory);
```

Java Example

The following example demonstrates setting the Add2IntsFactory's vol field to IMMUTABLE to tell Vertica it can cache the arguments and return value.

```
public class Add2IntsFactory extends ScalarFunctionFactory {

    @Override
    public void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes
returnType){
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface){
        return new Add2Ints();
    }
}
```

```
    }  
  
    // Class constructor  
    public Add2IntsFactory() {  
        // Tell Vertica that the same set of arguments will always result in the  
        // same return value.  
        vol = volatility.IMMUTABLE;  
    }  
}
```

Null Input Behavior

To indicate how your function reacts to NULL input, set the `strictness` field to one of the following values.

Value	Description
<code>CALLED_ON_NULL_INPUT</code>	The function must be called, even if one or more arguments are NULL.
<code>RETURN_NULL_ON_NULL_INPUT</code>	The function always returns a NULL value if any of its arguments are NULL.
<code>STRICT</code>	A synonym for <code>RETURN_NULL_ON_NULL_INPUT</code>
<code>DEFAULT_STRICTNESS</code>	The default strictness setting. This is the same as <code>CALLED_ON_NULL_INPUT</code> .

C++ Example

The following example demonstrates setting the null behavior of `Add2ints` so Vertica does not call the function with NULL values.

```
class Add2intsNullOnNullInputFactory : public Vertica::ScalarFunctionFactory  
{  
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)  
    { return vt_createFuncObj(srvInterface.allocator, Add2ints); }  
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,  
                             Vertica::ColumnTypes &argTypes,  
                             Vertica::ColumnTypes &returnType)  
    {  
        argTypes.addInt();  
        argTypes.addInt();  
        returnType.addInt();  
    }  
  
public:  
    Add2intsNullOnNullInputFactory() {strict = RETURN_NULL_ON_NULL_INPUT;}
```

```
};  
RegisterFactory(Add2intsNullOnNullInputFactory);
```

Java Example

The following example demonstrates setting the strictness setting of the Add2Ints example to STRICT. This means that if either of the two values to be added is NULL, Vertica can set the return value to NULL without having to call the Add2Ints function.

```
public class Add2IntsFactory extends ScalarFunctionFactory {  
  
    @Override  
    public void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes  
returnType){  
        argTypes.addInt();  
        argTypes.addInt();  
        returnType.addInt();  
    }  
  
    @Override  
    public ScalarFunction createScalarFunction(ServerInterface srvInterface){  
        return new Add2Ints();  
    }  
  
    public Add2IntsFactory() {  
        // Tell Vertica that any NULL arguments results in a NULL return value  
        strict = strictness.RETURN_NULL_ON_NULL_INPUT;  
    }  
}
```

C++API

This section provides APIs and examples for the C++ API for UDSFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

ScalarFunction and ScalarFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See [UDSF Class Overview](#) for general information about implementing the ScalarFunction and ScalarFunctionFactory classes.

ScalarFunction API

The API provides the following methods for extension by subclasses:

```
virtual void setup(ServerInterface &srvInterface,
                  const SizedColumnTypes &argTypes);

virtual void processBlock(ServerInterface &srvInterface,
                          BlockReader &arg_reader, BlockWriter &res_writer)=0;

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes);
```

ScalarFunctionFactory API

The API provides the following methods for extension by subclasses:

```
virtual ScalarFunction * createScalarFunction(ServerInterface &srvInterface)=0;

virtual void getPrototype(ServerInterface &srvInterface,
                          ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getReturnType(ServerInterface &srvInterface,
                            const SizedColumnTypes &argTypes, SizedColumnTypes &returnType);

virtual void getParameterType(ServerInterface &srvInterface,
                               SizedColumnTypes &parameterTypes);
```

C++ Example: Add2Ints

The following example shows a very basic subclass of `ScalarFunction` called `Add2ints`. As the name implies it adds two integers together, returning a single integer result. It also demonstrates including the main Vertica SDK header file (`Vertica.h`) and using the Vertica namespace. While not required, using the namespace saves you from having to prefix every Vertica SDK class reference with `Vertica::`.

ScalarFunction Implementation

```
// Include the top-level Vertica SDK file
#include "Vertica.h"
// Using the Vertica namespace means we don't have to prefix all
// class references with Vertica::
using namespace Vertica;
/*
 * ScalarFunction implementation for a UDSF that adds
 * two numbers together.
 */
class Add2Ints : public ScalarFunction
{
public:
/*
```

```
* This function does all of the actual processing for the UDF.
* In this case, it simply reads two integer values and returns
* their sum.
*
* The inputs are retrieved via arg_reader
* The outputs are returned via arg_writer
*/
virtual void processBlock(ServerInterface &srvInterface,
                          BlockReader &arg_reader,
                          BlockWriter &res_writer)
{
// While we have input to process
do {
// Read the two integer input parameters by calling the
// BlockReader.getIntRef class function
const vint a = arg_reader.getIntRef(0);
const vint b = arg_reader.getIntRef(1);
// Call BlockWriter.setInt to store the output value, which is the
// two input values added together
res_writer.setInt(a+b);
// Finish writing the row, and advance to the next output row
res_writer.next();
// Continue looping until there are no more input rows
} while (arg_reader.next());
}
};
```

ScalarFunctionFactory Implementation

```
/*
* This class provides metadata about the ScalarFunction class, and
* also instantiates a member of that class when needed.
*/
class Add2IntsFactory : public ScalarFunctionFactory
{
// return an instance of Add2Ints to perform the actual addition.
virtual ScalarFunction *createScalarFunction(ServerInterface &interface)
{
// Calls the vt_createFuncObj to create the new Add2Ints class instance.
return vt_createFuncObj(interface.allocator, Add2Ints);
}
// This function returns the description of the input and outputs of the
// Add2Ints class's processBlock function. It stores this information in
// two ColumnTypes objects, one for the input parameters, and one for
// the return value.
virtual void getPrototype(ServerInterface &interface,
                          ColumnTypes &argTypes,
                          ColumnTypes &returnType)
{
// Takes two ints as inputs, so add ints to the argTypes object
argTypes.addInt();
argTypes.addInt();
// returns a single int, so add a single int to the returnType object.
// Note that ScalarFunctions *always* return a single value.
returnType.addInt();
}
}
```

```
};
```

The RegisterFactory Macro

Use the `RegisterFactory` macro to register a `ScalarFunctionFactory` subclass. This macro instantiates the factory class and makes the metadata it contains available for Vertica to access. To call this macro, pass it the name of your factory class.

```
RegisterFactory(Add2IntsFactory);
```

C++ Example: Calling a UDSF from a Check Constraint

This example shows you the C++ code needed to create a UDSF that can be called by a check constraint. The name of the sample function is `LargestSquareBelow`. The sample function determines the largest number whose square is less than the number in the subject column. For example, if the number in the column is 1000, the largest number whose square is less than 1000 is 31 (961).

Important: A UDSF used within a check constraint must be immutable, and the constraint must handle null values properly. Otherwise, the check constraint might not work as you intended. In addition, Vertica evaluates the predicate of an enabled check constraint on every row that is loaded or updated, so consider performance in writing your function.

For information on check constraints, see [Check Constraints](#) the Administrator's Guide.

For information about implementing the `ScalarFunction` and `ScalarFunctionFactory` classes, see [UDSF Class Overview](#).

Loading and Using the Example

The following example shows how you can create and load a library named `MySqlLib`, using `CREATE LIBRARY`. Adjust the library path in this example to the absolute path and file name for the location where you saved the shared object `LargestSquareBelow`.

```
=> CREATE OR REPLACE LIBRARY MySqlLib AS '/home/dbadmin/LargestSquareBelow.so';
```

1. After you create and load the library, add the function to the catalog using the `CREATE FUNCTION (UDF)` statement.

```
=> CREATE OR REPLACE FUNCTION largestSqBelow AS LANGUAGE 'C++' NAME 'LargestSquareBelowInfo' LIBRARY
MySqlLib;
```

2. Next, include the UDSF in a check constraint.

```
=> CREATE TABLE squaretest(
  ceiling INTEGER UNIQUE,
  CONSTRAINT chk_sq CHECK (largestSqBelow(ceiling) < ceiling*ceiling)
);
```

3. Add data to the table, squaretest.

```
=> COPY squaretest FROM stdin DELIMITER ',' NULL 'null';
-1
null
0
1
1000
1000000
1000001
\.
```

Your output should be similar to the following sample, based upon the data you use:

```
SELECT ceiling, largestSqBelow(ceiling)
FROM squaretest ORDER BY ceiling;
```

ceiling	largestSqBelow
-1	
0	
1	0
1000	31
1000000	999
1000001	1000

(7 rows)

ScalarFunction Implementation

This `ScalarFunction` implementation does the processing work for a UDSF that determines the largest number whose square is less than the number input.

```
#include "Vertica.h"
/*
 * ScalarFunction implementation for a UDSF that
 * determines the largest number whose square is less than
 * the number input.
 */
class LargestSquareBelow : public Vertica::ScalarFunction
{
```

```
public:
/*
 * This function does all of the actual processing for the UDSF.
 * The inputs are retrieved via arg_reader
 * The outputs are returned via arg_writer
 *
 */
virtual void processBlock(Vertica::ServerInterface &srvInterface,
                          Vertica::BlockReader &arg_reader,
                          Vertica::BlockWriter &res_writer)
{
  if (arg_reader.getNumCols() != 1)
    vt_report_error(0, "Function only accept 1 argument, but %zu provided", arg_reader.getNumCols());
  // While we have input to process
  do {
    // Read the input parameter by calling the
    // BlockReader.getIntRef class function
    const Vertica::vint a = arg_reader.getIntRef(0);
    Vertica::vint res;
    //Determine the largest square below the number
    if ((a != Vertica::vint_null) && (a > 0))
    {
      res = (Vertica::vint)sqrt(a - 1);
    }
    else
      res = Vertica::vint_null;
    //Call BlockWriter.setInt to store the output value,
    //which is the largest square
    res_writer.setInt(res);
    //Write the row and advance to the next output row
    res_writer.next();
    //Continue looping until there are no more input rows
  } while (arg_reader.next());
}
};
```

ScalarFunctionFactory Implementation

This `ScalarFunctionFactory` implementation does the work of handling input and output, and marks the function as immutable (a requirement if you plan to use the UDSF within a check constraint).

```
class LargestSquareBelowInfo : public Vertica::ScalarFunctionFactory
{
  //return an instance of LargestSquareBelow to perform the computation.
  virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
  //Call the vt_createFuncObj to create the new LargestSquareBelow class instance.
  { return Vertica::vt_createFuncObject<LargestSquareBelow>(srvInterface.allocator); }

  /*
   * This function returns the description of the input and outputs of the
   * LargestSquareBelow class's processBlock function. It stores this information in
   * two ColumnTypes objects, one for the input parameter, and one for
   * the return value.
   */
};
```

```
virtual void getPrototype(Vertica::ServerInterface &srvInterface,  
                        Vertica::ColumnTypes &argTypes,  
                        Vertica::ColumnTypes &returnType)  
{  
    // Takes one int as input, so adds int to the argTypes object  
    argTypes.addInt();  
    // Returns a single int, so add a single int to the returnType object.  
    // ScalarFunctions always return a single value.  
    returnType.addInt();  
}  
public:  
    // the function cannot be called within a check constraint unless the UDX author  
    // certifies that the function is immutable:  
    LargestSquareBelowInfo() { vol = Vertica::IMMUTABLE; }  
};
```

The RegisterFactory Macro

Use the `RegisterFactory` macro to register a `ScalarFunctionFactory` subclass. This macro instantiates the factory class and makes the metadata it contains available for Vertica to access. To call this macro, pass it the name of your factory class.

```
RegisterFactory(LargestSquareBelowInfo);
```

Java API

This section provides APIs and examples for the Java API for UDSFs.

For information on setting up a Java development environment and compiling and packaging libraries, see [Developing with the Java SDK](#).

ScalarFunction and ScalarFunctionFactory Java Interface

This section describes information that is specific to the Java API. See [UDSF Class Overview](#) for general information about implementing the `ScalarFunction` and `ScalarFunctionFactory` classes.

ScalarFunction API

The API provides the following methods for extension by subclasses:

```
void setup(ServerInterface srvInterface, SizedColumnTypes argTypes);  
  
abstract void processBlock(ServerInterface srvInterface, BlockReader arg_reader,
```

```
BlockWriter res_writer) throws UdfException, DestroyInvocation;  
void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes);
```

ScalarFunctionFactory API

The API provides the following methods for extension by subclasses:

```
abstract ScalarFunction createScalarFunction(ServerInterface srvInterface);  
  
abstract void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType);  
  
void getReturnType(ServerInterface srvInterface, SizedColumnTypes argTypes,  
    SizedColumnTypes returnType) throws UdfException;  
  
void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);
```

Java Example: Add2Ints

The Add2Ints scalar function adds two integers together, returning a single integer result.

Loading and Using the Example

Use [CREATE LIBRARY](#) to load the jar file containing the function, and then use [CREATE FUNCTION \(UDF\)](#) to declare the function as in the following example:

```
=> CREATE FUNCTION add2ints AS LANGUAGE 'Java' NAME  
    'com.mycompany.example.Add2intsFactory' LIBRARY add2intslib;
```

The following example shows how to use this function.

```
=> SELECT Add2Ints(27,15);  
Add2ints  
-----  
      42  
(1 row)  
=> SELECT * FROM MyTable;  
 a | b  
-----+-----  
  7 |  0  
 12 |  2  
 12 |  6  
 18 |  9  
  1 |  1  
 58 |  4  
450 | 15
```

```
(7 rows)
=> SELECT * FROM MyTable WHERE Add2ints(a, b) > 20;
  a | b
-----+-----
  18 | 9
  58 | 4
 450 | 15
(3 rows)
```

Implementation

The following code example is the full source of the Java Add2ints example. To simplify handling the source code, the `ScalarFunction` class is defined as an inner class of the `ScalarFunctionFactory` class.

```
// You will need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.example;
// Import all of the Vertica SDK
import com.vertica.sdk.*;
public class Add2intsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
                            ColumnTypes argTypes,
                            ColumnTypes returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    // This ScalarFunction is defined as an inner class of
    // its ScalarFunctionFactory class. This gets around having
    // to have a separate source file for this public class.
    public class Add2ints extends ScalarFunction
    {
        @Override
        public void processBlock(ServerInterface srvInterface,
                                BlockReader argReader,
                                BlockWriter resWriter)
            throws UdfException, DestroyInvocation
        {
            do {
                // The input and output objects have already loaded
                // the first row, so you can start reading and writing
                // values immediately.

                // Get the two integer arguments from the BlockReader
                long a = argReader.getLong(0);
                long b = argReader.getLong(1);

                // Process the arguments and come up with a result. For this
                // example, just add the two arguments together.
                long result = a+b;
            } while (argReader.next());
        }
    }
}
```

```
// Write the integer output value.
resWriter.setLong(result);

// Advance the output BlockWriter to the next row.
resWriter.next();

// Continue processing input rows until there are no more.
} while (argReader.next());
}
}

@Override
public ScalarFunction createScalarFunction(ServerInterface srvInterface)
{
    return new Add2ints();
}
```

Python API

This section provides APIs and examples for the Python SDK for UDSFs.

For more information on setting up a Python development environment, see [Developing with the Python SDK](#).

Scalar Function and Scalar Function Factory Python Interface

This section describes information that is specific to the Python API. See [UDSF Class Overview](#) for general information about implementing the `ScalarFunction` and `ScalarFunctionFactory` classes.

ScalarFunction API

The API provides the following methods for extension by subclasses:

```
class python_udx(vertica_sdk.ScalarFunctionFactory):

    def __init__(self):
        pass
    def setup(self, server_interface, col_types):
        pass
    def processBlock(self, server_interface, block_reader, block_writer):
        pass
    def destroy(self, server_interface, col_types):
        pass
```

ScalarFunctionFactory API

The API provides the following methods for extension by subclasses:

```
class python_udx_factory(vertica_sdk.ScalarFunctionFactory):  
  
    def createScalarFunction(self, srv):  
        pass  
    def getPrototype(self, srv_interface, arg_types, return_type):  
        pass  
    def getReturnType(self, srv_interface, arg_types, return_type):  
        pass
```

Python Example: add2ints

The `add2ints` scalar function adds two integers together, returning a single integer result.

You can find more UDX examples in the VerticaGithub repository,
<https://github.com/vertica/UDx-Examples>.

UDSF Python Code

```
import vertica_sdk  
  
class add2ints(vertica_sdk.ScalarFunction):  
    """Return the sum of two integer columns"""  
  
    def __init__(self):  
        pass  
  
    def setup(self, server_interface, col_types):  
        pass  
  
    def processBlock(self, server_interface, arg_reader, res_writer):  
        # Writes a string to the UDX log file.  
        server_interface.log("Python UDX - Adding 2 ints!")  
        while(True):  
            # Example of error checking best practices.  
            if arg_reader.isNull(0) or arg_reader.isNull(1):  
                raise ValueError("I found a NULL!")  
            else:  
                first_int = arg_reader.getInt(0) # Input column  
                second_int = arg_reader.getInt(1) # Input column  
                res_writer.setInt(first_int + second_int) # Sum of input columns.  
                res_writer.next() # Read the next row.  
            if not arg_reader.next():  
                # Stop processing when there are no more input rows.  
                break  
  
    def destroy(self, server_interface, col_types):
```

```
pass

class add2ints_factory(vertica_sdk.ScalarFunctionFactory):

    def createScalarFunction(self, srv):
        return add2ints()

    def getPrototype(self, srv_interface, arg_types, return_type):
        arg_types.addInt()
        arg_types.addInt()
        return_type.addInt()

    def getReturnType(self, srv_interface, arg_types, return_type):
        return_type.addInt()
```

Load the Function and Library

Create the library and the function.

```
=> CREATE LIBRARY pylib AS '/home/dbadmin/python_udx/add2ints/add2ints.py' LANGUAGE 'Python';
CREATE LIBRARY
=> CREATE FUNCTION add2ints AS LANGUAGE 'Python' NAME 'add2ints_factory' LIBRARY pylib fenced;
CREATE FUNCTION
```

Querying Data with the Function

The following shows how to run a query with the UDSF.

```
=> SELECT add2ints(10, 10);
   add2ints
-----
         20
(1 row)
```

You can query on a table with two integer columns as follows:

```
=> SELECT numbs_1, numbs_2, add2ints(numbs_1, numbs_2) AS add2ints_sum
   FROM bunch_of_numbers;
 numbs_1 | numbs_2 | add2ints_sum
-----+-----+-----
      10 |       10 |           20
       1 |        4 |            5
        6 |        6 |           12
      30 |      144 |          174
(4 rows)
```

Python Example: currency_convert

The `currency_convert` scalar function reads two values from a table, a currency and a value. It then converts the item's value to USD, returning a single float result.

You can find more UDX examples in the VerticaGithub repository, <https://github.com/vertica/UDx-Examples>.

UDSF Python Code

```
import vertica_sdk
import decimal

rates2USD = {'USD': 1.000,
             'EUR': 0.89977,
             'GBP': 0.68452,
             'INR': 67.0345,
             'AUD': 1.39187,
             'CAD': 1.30335,
             'ZAR': 15.7181,
             'XXX': -1.0000}

class currency_convert(vertica_sdk.ScalarFunction):
    """Converts a money column to another currency

    Returns a value in USD.

    """
    def __init__(self):
        pass

    def setup(self, server_interface, col_types):
        pass

    def processBlock(self, server_interface, block_reader, block_writer):
        while(True):
            currency = block_reader.getString(0)
            try:
                rate = decimal.Decimal(rates2USD[currency])

            except KeyError:
                server_interface.log("ERROR: {} not in dictionary.".format(currency))
                # Scalar functions always need a value to move forward to the
                # next input row. Therefore, we need to assign it a value to
                # move beyond the error.
                currency = 'XXX'
                rate = decimal.Decimal(rates2USD[currency])

            starting_value = block_reader.getNumeric(1)
            converted_value = decimal.Decimal(starting_value / rate)
            block_writer.setNumeric(converted_value)
            block_writer.next()
            if not block_reader.next():
```

```
        break

    def destroy(self, server_interface, col_types):
        pass

class currency_convert_factory(vertica_sdk.ScalarFunctionFactory):

    def createScalarFunction(self, srv):
        return currency_convert()

    def getPrototype(self, srv_interface, arg_types, return_type):
        arg_types.addVarchar()
        arg_types.addNumeric()
        return_type.addNumeric()

    def getReturnType(self, srv_interface, arg_types, return_type):
        return_type.addNumeric(9,4)
```

Load the Function and Library

Create the library and the function.

```
=> CREATE LIBRARY pylib AS '/home/dbadmin/python_udx/currency_convert/currency_convert.py' LANGUAGE
'Python';
CREATE LIBRARY
=> CREATE FUNCTION currency_convert AS LANGUAGE 'Python' NAME 'currency_convert_factory' LIBRARY
pylib fenced;
CREATE FUNCTION
```

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

```
=> SELECT product, currency_convert(currency, value) AS cost_in_usd
FROM items;
 product | cost_in_usd
-----+-----
Shoes    | 133.4008
Soccer Ball | 110.2817
Coffee   | 13.5190
Surfboard | 176.2593
Hockey Stick | 76.7177
Car      | 17000.0000
Software | 10.4424
Hamburger | 7.5000
Fish     | 130.4272
Cattle   | 269.2367
(10 rows)
```

Python Example: validate_url

The `validate_url` scalar function reads a string from a table, a URL. It then validates if the URL is responsive, returning a status code or a string indicating the attempt failed.

You can find more UDX examples in the VerticaGithub repository, <https://github.com/vertica/UDx-Examples>.

UDSF Python Code

```
import vertica_sdk
import urllib.request
import time

class validate_url(vertica_sdk.ScalarFunction):
    """Validates HTTP requests.

    Returns the status code of a webpage. Pages that cannot be accessed return
    "Failed to load page."

    """

    def __init__(self):
        pass

    def setup(self, server_interface, col_types):
        pass

    def processBlock(self, server_interface, arg_reader, res_writer):
        # Writes a string to the UDX log file.
        server_interface.log("Validating webpage accessibility - UDX")

        while(True):
            url = arg_reader.getString(0)
            try:
                status = urllib.request.urlopen(url).getcode()
                # Avoid overwhelming web servers -- be nice.
                time.sleep(2)
            except (ValueError, urllib.error.HTTPError, urllib.error.URLError):
                status = 'Failed to load page'
                res_writer.setString(str(status))
                res_writer.next()
            if not arg_reader.next():
                # Stop processing when there are no more input rows.
                break

    def destroy(self, server_interface, col_types):
        pass

class validate_url_factory(vertica_sdk.ScalarFunctionFactory):

    def createScalarFunction(self, srv):
        return validate_url()
```

```
def getPrototype(self, srv_interface, arg_types, return_type):
    arg_types.addVarchar()
    return_type.addChar()

def getReturnType(self, srv_interface, arg_types, return_type):
    return_type.addChar(20)
```

Load the Function and Library

Create the library and the function.

```
=> CREATE OR REPLACE LIBRARY pylib AS 'webpage_tester/validate_url.py' LANGUAGE 'Python';
=> CREATE OR REPLACE FUNCTION validate_url AS LANGUAGE 'Python' NAME 'validate_url_factory' LIBRARY
pylib fenced;
```

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

```
=> SELECT url, validate_url(url) AS url_status FROM webpages;
      url                                     | url_status
-----+-----
http://my.vertica.com/documentation/vertica/ | 200
http://www.google.com/                       | 200
http://www.mass.gov.com/                     | Failed to load page
http://www.espn.com                          | 200
http://blah.blah.blah.blah                  | Failed to load page
http://www.hpe.com/                          | 200
(6 rows)
```

R API

This section provides APIs and examples for the R API for UDSFs.

For information on setting up an R development environment and compiling and packaging libraries, see [Developing with the R SDK](#).

Scalar Function and Scalar Function Factory R Interface

This section describes information that is specific to the R API. See [R SDK API Documentation](#) for general information about implementing Scalar functions.

Scalar Function API

The API provides the following framework for extension by R function:

```
FunctionName <- function(input.data.frame, parameters.data.frame) {  
  # Computations  
  
  # The function must return a data frame.  
  return(output.data.frame)  
}
```

Scalar Function Factory API

The API provides the following framework for extension by R function:

```
FunctionNameFactory <- function() {  
  list(name = FunctionName,  
        udxtype = c("scalar"),  
        intype = c("int"),  
        outtype = c("int"))  
}
```

For a complete list of factory options, see the R API documentation for [Factory Function](#).

R Example: SalesTaxCalculator

The `SalesTaxCalculator` scalar function reads a float and a varchar from a table, an item's price and the state abbreviation. It then uses the state abbreviation to find the sales tax rate from a list and calculates the item's price including the state's sales tax, returning the total cost of the item.

You can find more UDX examples in the VerticaGithub repository, <https://github.com/vertica/UDx-Examples>.

Load the Function and Library

Create the library and the function.

```
=> CREATE OR REPLACE LIBRARY rLib AS 'sales_tax_calculator.R' LANGUAGE 'R';  
CREATE LIBRARY  
=> CREATE OR REPLACE FUNCTION SalesTaxCalculator AS LANGUAGE 'R' NAME 'SalesTaxCalculatorFactory'  
LIBRARY rLib FENCED;  
CREATE FUNCTION
```

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

```
=> SELECT item, state_abbreviation,
         price, SalesTaxCalculator(price, state_abbreviation) AS Price_With_Sales_Tax
FROM inventory;
```

item	state_abbreviation	price	Price_With_Sales_Tax
Scarf	AZ	6.88	7.53016
Software	MA	88.31	96.655295
Soccer Ball	MS	12.55	13.735975
Beads	LA	0.99	1.083555
Baseball	TN	42.42	46.42869
Cheese	WI	20.77	22.732765
Coffee Mug	MA	8.99	9.839555
Shoes	TN	23.99	26.257055

(8 rows)

UDSF R Code

```
SalesTaxCalculator <- function(input.data.frame) {
  # Not a complete list of states in the USA, but enough to get the idea.
  state.sales.tax <- list(ma = 0.0625,
                        az = 0.087,
                        la = 0.0891,
                        tn = 0.0945,
                        wi = 0.0543,
                        ms = 0.0707)
  for (state_abbreviation in input.data.frame[, 2]) {
    # Ensure state abbreviations are lowercase.
    lower_state <- tolower(state_abbreviation)
    # Check if the state is in our state.sales.tax list.
    if (is.null(state.sales.tax[[lower_state]])) {
      stop("State is not in our small sample!")
    } else {
      sales.tax.rate <- state.sales.tax[[lower_state]]
      item.price <- input.data.frame[, 1]
      # Calculate the price including sales tax.
      price.with.sales.tax <- (item.price) + (item.price * sales.tax.rate)
    }
  }
  return(price.with.sales.tax)
}

SalesTaxCalculatorFactory <- function() {
  list(name = SalesTaxCalculator,
       udxtype = c("scalar"),
       intype = c("float", "varchar"),
       outtype = c("float"))
}
```

R Example: kmeans

The `KMeans_User` scalar function reads any number of columns from a table, the observations. It then uses the observations and the two parameters when applying the kmeans clustering algorithm to the data, returning an integer value associated with the cluster of the row.

You can find more UDX examples in the VerticaGithub repository, <https://github.com/vertica/UDx-Examples>.

Load the Function and Library

Create the library and the function.

```
=> CREATE OR REPLACE LIBRARY rLib AS 'kmeans.R' LANGUAGE 'R';  
CREATE LIBRARY  
=> CREATE OR REPLACE FUNCTION KMeans_User AS LANGUAGE 'R' NAME 'KMeans_UserFactory' LIBRARY rLib  
FENCED;  
CREATE FUNCTION
```

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

```
=> SELECT spec,  
        KMeans_User(s1, sw, p1, pw USING PARAMETERS clusters = 3, nstart = 20)  
FROM iris;  
   spec | KMeans_User  
-----+-----  
Iris-setosa | 2  
.  
.  
.  
(150 rows)
```

UDSF R Code

```
KMeans_User <- function(input.data.frame, parameters.data.frame) {  
  # Take the clusters and nstart parameters passed by the user and assign them  
  # to variables in the function.  
  if ( is.null(parameters.data.frame[["clusters"]]) ) {  
    stop("NULL value for clusters! clusters cannot be NULL.")  
  } else {  
    clusters.value <- parameters.data.frame[["clusters"]]  
  }  
  if ( is.null(parameters.data.frame[["nstart"]]) ) {  
    stop("NULL value for nstart! nstart cannot be NULL.")  
  } else {  
    nstart.value <- parameters.data.frame[["nstart"]]  
  }  
  # Apply the algorithm to the data.  
  kmeans.clusters <- kmeans(input.data.frame[, 1:length(input.data.frame)],  
    clusters.value, nstart = nstart.value)  
  final.output <- data.frame(kmeans.clusters$cluster)  
  return(final.output)  
}  
  
KMeans_UserFactory <- function() {  
  list(name = KMeans_User,  
    udxtype = c("scalar"),  
    # Since this is a polymorphic function the intype must be any  
    intype = c("any"),  
    outtype = c("int"),  
    parameterstypecallback=KMeansParameters)  
}  
  
KMeansParameters <- function() {  
  parameters <- list(datatype = c("int", "int"),  
    length = c("NA", "NA"),  
    scale = c("NA", "NA"),  
    name = c("clusters", "nstart"))  
  return(parameters)  
}
```

Transform Functions (UDTFs)

A User-Defined Transform Function (UDTF) lets you transform a table of data into another table. It reads one or more arguments (treated as a row of data), and returns zero or more rows of data consisting of one or more columns. A UDTF can produce any number of rows as output. However, each row it outputs must be complete. Advancing to the next row without having added a value for each column produces incorrect results.

The schema of the output table does not need to correspond to the schema of the input table—they can be totally different. The UDTF can return any number of output rows for each row of input.

UDTFs can only be used in the [SELECT](#) list that contains just the UDTF call and a required `OVER` clause. A multi-phase UDTF can make use of partition columns (`PARTITION BY`), but other UDTFs cannot.

UDTFs are run after `GROUP BY`, but before the final `ORDER BY`, when used in conjunction with `GROUP BY` and `ORDER BY` in a statement. The `ORDER BY` clause may contain only columns or expressions that are in a window partition clause (see [Window Partitioning](#)).

UDTFs can take up to 1600 parameters (input columns). Attempting to pass more parameters to a UDTF results in an error.

UDTF Class Overview

You create your UDTF by subclassing two classes defined by the Vertica SDK: `TransformFunction` and `TransformFunctionFactory`.

The `TransformFunctionFactory` performs two roles:

- It provides the number of parameters and their data types accepted by the UDTF and the number of output columns and their data types UDTF's output. Vertica uses this data when you call the [CREATE TRANSFORM FUNCTION](#) SQL statement to add the function to the database catalog.
- It returns an instance of the UDTF function's `TransformFunction` subclass that Vertica can call to process data.

TransformFunction

The `TransformFunction` class is where you perform the data-processing, transforming input rows into output rows. Your subclass must define the `processPartition()` method. It may define methods to set up and tear down the function.

Performing the Transformation

The `processPartition()` method carries out all of the processing that you want your UDTF to perform. When a user calls your function in a SQL statement, Vertica bundles together the data from the function parameters and passes it to `processPartition()`.

The input and output of the `processPartition()` method are supplied by objects of the `BlockReader` and `BlockWriter` classes. They define methods that you use to read the input data and write the output data for your UDTF.

Your `processPartition()` method should follow this basic pattern:

- Extract the input parameters by calling a data-type-specific function in the `PartitionReader` object to extract each input parameter. All of these functions take a single parameter: the column number in the input row that you want to read. Your function might need to handle NULL values.
- Perform the actual transformation.
- Write the output, if any. Unlike a UDSF, outputting data is optional for a UDTF. However, if it does write output, it must supply values for all of the output columns you defined in your factory. Similarly to reading input columns, the `PartitionWriter` object has functions for writing each type of data to the output row.
- Advance to the next row by calling `ProcessReader.next()`. This function returns true if there is another row of input data to process and false if all the data in the partition has been read. Once the input rows are exhausted, your UDTF exits so its results are returned back to Vertica

Note: In some cases, you may want to determine the number and types of parameters using `PartitionReader`'s `getNumCols()` and `getTypeMetaData()` functions, instead of just hard-coding the data types of the columns in the input row. This is useful if you want your `TransformFunction` to be able to process input tables with different schemas. You can then use different `TransformFunctionFactory` classes to define multiple function signatures that call the same `TransformFunction` class. See [Handling](#)

[Different Numbers and Types of Arguments](#) for more information.

Setting Up and Tearing Down

The `TransformFunction` class defines two additional methods that you can optionally implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDX API (see [Allocating Resources for UDXs](#) for details).

TransformFunctionFactory

The `TransformFunctionFactory` class tells Vertica metadata about your UDTF: its number of parameters and their data types, as well as the data type of its return value. It also instantiates a subclass of `TransformFunction`.

You must implement the following methods in your `TransformFunctionFactory`:

- `getPrototype()` returns two `ColumnTypes` objects that describe the columns your UDTF takes as input and returns as output.
- `createTransformFunction()` instantiates your `TransformFunction` subclass.
- `getReturnType()` tells Vertica details about the output values: the width of variable-sized data types (such as `VARCHAR`) and the precision of data types that have settable precision (such as `TIMESTAMP`). You can also set the names of the output columns using this function.

Note: The `getReturnType()` function is required for UDTFs. It is optional for UDXs that return single values, such as User-Defined Scalar Functions.

See Also

[Creating Multi-Phase UDTFs](#)

Creating Multi-Phase UDTFs

Multi-phase UDTFs let you break your data processing into multiple steps. Using this feature, your UDTFs can perform processing in a way similar to Hadoop or other MapReduce frameworks. You can use the first phase to break down and gather data, and then use subsequent phases to process the data. For example, the first phase of your UDTF could extract

specific types of user interactions from a web server log stored in the column of a table, and subsequent phases could perform analysis on those interactions.

Multi-phase UDTFs also let you decide where processing should occur: locally on each node, or throughout the cluster. If your multi-phase UDTF is like a MapReduce process, you want the first phase of your multi-phase UDTF to process data that is stored locally on the node where the instance of the UDTF is running. This prevents large segments of data from being copied around the Vertica cluster. Depending on the type of processing being performed in later phases, you may choose to have the data segmented and distributed across the Vertica cluster.

Each phase of the UDTF is the same as a traditional (single-phase) UDTF: it receives a table as input, and generates a table as output. The schema for each phase's output does not have to match its input, and each phase can output as many or as few rows as it wants.

You create a subclass of `TransformFunction` to define the processing performed by each stage. If you already have a `TransformFunction` from a single-phase UDTF that performs the processing you want a phase of your multi-phase UDTF to perform, you can easily adapt it to work within the multi-phase UDTF.

What makes a multi-phase UDTF different from a traditional UDTF is the factory class you use. You define a multi-phase UDTF using a subclass of `MultiPhaseTransformFunctionFactory`, rather than the `TransformFunctionFactory`. This special factory class acts as a container for all of the phases in your multi-step UDTF. It provides Vertica with the input and output requirements of the entire multi-phase UDTF (through the `getPrototype()` function), and a list of all the phases in the UDTF.

Within your subclass of the `MultiPhaseTransformFunctionFactory` class, you define one or more subclasses of `TransformFunctionPhase`. These classes fill the same role as the `TransformFunctionFactory` class for each phase in your multi-phase UDTF. They define the input and output of each phase and create instances of their associated `TransformFunction` classes to perform the processing for each phase of the UDTF. In addition to these subclasses, your `MultiPhaseTransformFunctionFactory` includes fields that provide a handle to an instance of each of the `TransformFunctionPhase` subclasses.

Partitioning Options for Processing Local Data

UDTFs typically process data that is partitioned in a specific way. For example, a UDTF that processes a web server log file to count the number of hits referred by each partner web site needs to have its input partitioned by a referrer column. Each instance of the UDTF sees the hits referred by a particular partner site so it can count them.

In cases like this, the [window partitioning clause](#) should use a `PARTITION BY` clause. Each node in the cluster partitions the data it stores, sends some of these partitions off to other

nodes, and then consolidates the partitions it receives from other nodes and runs an instance of the UDTF to process them.

In other cases, a UDTF might not need to partition input data in a particular way—for example, a UDTF that parses data out of an Apache log file. In this case, you can specify that each UDTF instance process only the data that is stored locally by the node on which it is running. By eliminating the overhead of partitioning data, processing can be much more efficient.

You can tell a UDTF to process only local data with one of the following window partitioning options:

- **PARTITION BEST:** For thread-safe UDTFs only, optimizes performance through multi-threaded queries across multiple nodes.
- **PARTITION NODES:** Optimizes performance of single-threaded queries across multiple nodes.

The query must specify a source table that is replicated across all nodes and contains a single row (similar to the [DUAL](#) table). For example, the following statements call a UDTF that parses locally-stored Apache log files:

```
=> CREATE TABLE rep (dummy INTEGER) UNSEGMENTED ALL NODES;
CREATE TABLE
=> INSERT INTO rep VALUES (1);
OUTPUT
-----
      1
(1 row)
=> SELECT ParseLogFile('/data/apache/log*') OVER (PARTITION BEST) FROM rep;
```

C++ API

This section provides APIs and examples for the C++ API for UDTFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

TransformFunction and TransformFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See [UDTF Class Overview](#) for general information about implementing the `TransformFunction` and `TransformFunctionFactory` classes.

TransformFunction API

The API provides the following methods for extension by subclasses:

```
virtual void setup(ServerInterface &srvInterface,  
                  const SizedColumnTypes &argTypes);  
  
virtual void processPartition(ServerInterface &srvInterface,  
                              PartitionReader &input_reader, PartitionWriter &output_writer)=0;  
  
virtual void destroy(ServerInterface &srvInterface,  
                     const SizedColumnTypes &argTypes);
```

TransformFunctionFactory API

The API provides the following methods for extension by subclasses:

```
virtual TransformFunction *  
  createTransformFunction (ServerInterface &srvInterface)=0;  
  
virtual void getPrototype(ServerInterface &srvInterface,  
                          ColumnTypes &argTypes, ColumnTypes &returnType)=0;  
  
virtual void getReturnType(ServerInterface &srvInterface,  
                           const SizedColumnTypes &argTypes,  
                           SizedColumnTypes &returnType)=0;  
  
virtual void getParameterType(ServerInterface &srvInterface,  
                              SizedColumnTypes &parameterTypes);
```

MultiPhaseTransformFunctionFactory API

The `MultiPhaseTransformFunctionFactory` class extends `TransformFunctionFactory`. The API provides the following additional methods for extension by subclasses:

```
virtual void getPhases(ServerInterface &srvInterface,  
                       std::vector< TransformFunctionPhase * > &phases)=0;
```

If using this factory you must also extend `TransformFunctionPhase`. See the SDK reference documentation.

C++ Example: String Tokenizer

The following example shows a subclass of `TransformFunction` named `StringTokenizer`. It defines a UDTF that reads a table containing an `INTEGER` ID column and a `VARCHAR` column. It breaks the text in the `VARCHAR` column into tokens (individual words). It returns a table containing each token, the row it occurred in, and its position within the string.

Loading and Using the Example

The following example shows how to load the function into Vertica. It assumes that the `TransformFunctions.so` library that contains the function has been copied to the `dbadmin` user's home directory on the initiator node.

```
=> CREATE LIBRARY TransformFunctions AS
    '/home/dbadmin/TransformFunctions.so';
CREATE LIBRARY
=> CREATE TRANSFORM FUNCTION tokenize
    AS LANGUAGE 'C++' NAME 'TokenFactory' LIBRARY TransformFunctions;
CREATE TRANSFORM FUNCTION
```

You can then use it from SQL statements, for example:

```
=> CREATE TABLE T (url varchar(30), description varchar(2000));
CREATE TABLE
=> INSERT INTO T VALUES ('www.amazon.com','Online retail merchant and provider of cloud services');
OUTPUT
-----
      1
(1 row)
=> INSERT INTO T VALUES ('www.hp.com','Leading provider of computer hardware and imaging solutions');
OUTPUT
-----
      1
(1 row)
=> INSERT INTO T VALUES ('www.vertica.com','World's fastest analytic database');
OUTPUT
-----
      1
(1 row)
=> COMMIT;
COMMIT

=> -- Invoke the UDT
=> SELECT url, tokenize(description) OVER (partition by url) FROM T;
      url      | words
-----+-----
www.amazon.com | Online
www.amazon.com | retail
www.amazon.com | merchant
www.amazon.com | and
www.amazon.com | provider
```

```
www.amazon.com | of
www.amazon.com | c
www.amazon.com | loud
www.amazon.com | services
www.hp.com | Leading
www.hp.com | provider
www.hp.com | of
www.hp.com | computer
www.hp.com | hardware
www.hp.com | and
www.hp.com | im
www.hp.com | aging
www.hp.com | solutions
www.vertica.com | World's
www.vertica.com | fastest
www.vertica.com | analytic
www.vertica.com | database
(22 rows)
```

Notice that the number of rows and columns in the result table are different than the input table. This is one of the strengths of a UDTF.

TransformFunction Implementation

The following code defines the `StringTokenizer` class.

```
#include "Vertica.h"
#include <sstream>
// Use the Vertica namespace to make referring
// to SDK classes easier.
using namespace Vertica;
using namespace std;

// The primary class for the StringTokenizer UDTF. This does the actual work.
class StringTokenizer : public TransformFunction {
    // Called for each partition in the table.
    virtual void processPartition(ServerInterface &srvInterface,
        PartitionReader &inputReader,
        PartitionWriter &outputWriter) {
        // Use a top-level try to prevent exceptions from bubbling up.
        try {
            // Loop through the input rows
            do {
                // First check if the id is NULL. If so, skip the row without
                // writing any output.
                if (inputReader.isNull(0))
                {
                    srvInterface.log("Skipping row with NULL ID");
                    continue;
                }
                // Get an input row containing an int and a varchar
                const vint row = inputReader.getIntRef(0);
                const VString &sentence = inputReader.getStringRef(1);
                // If input string is NULL, then output NULL tokens and
                // positions.
                if (sentence.isNull())
```

```
{
  outputWriter.setNull(0);
  outputWriter.setInt(1,row);
  outputWriter.setNull(2);
  // Move on to the next row
  outputWriter.next();
}
else
{
  // Tokenize the string and output the words
  std::string tmp = sentence.str();
  istringstream ss(tmp);
  int wordcount = 0; // Track which word we're on
  do
  {
    std::string buffer;
    ss >> buffer;

    // Copy to output
    if (!buffer.empty()) {
      // To write a string, need to get the buffer to
      // write to, then copy the token into it.
      VString &word = outputWriter.getStringRef(0);
      word.copy(buffer);
      // 2nd column is row id of input
      outputWriter.setInt(1,row);
      // 3rd column is the token's position in the row
      outputWriter.setInt(2,wordcount++);

      // Move on to the next output row.
      outputWriter.next();
    }
  } while (ss);
} while (inputReader.next()); // Get next row (if any)
} catch (exception& e) {
  // Standard exception. Report the error.
  vt_report_error(0, "Exception while processing partition: [%s]",
    e.what());
}
};
```

The `processPartition()` function in this example follows a pattern that you will follow in your own UDTF: it loops over all rows in the table partition that Vertica sends it, processing each row. For UDTFs you do not have to actually process every row. You can exit your function without having read all of the input without any issues. You may choose to do this if your UDTF is performing some sort search or some other operation where it can determine that the rest of the input is unneeded.

In this example, `processPartition()` first extracts the ID number of the row. Then it extracts the `VString` containing the text from the `PartitionReader` object. The `VString` class represents a Vertica string value (`VARCHAR` or `CHAR`).

When developing UDTFs, you often need to handle NULL input values in a special way. This example has two different reactions to a NULL value:

- If the ID column is NULL, then `processPartition()` skips the row and moves on to the next row. It writes a message to the Vertica log to indicate that it is skipping the row.
- If the VARCHAR column is NULL, `processPartition()` outputs a single row that contains the `row_id` value and NULLs for both the `token` and `token_position` columns.

After handling any NULL values, the `processPartition()` function shown in the example moves on to performing the actual processing. It breaks the string into individual tokens and outputs a row for each token.

Similarly to reading input columns, the `PartitionWriter` object has functions for writing each type of data to the output row. In this case, the example:

- Calls the `PartitionWriter` object's `getStringRef()` function to allocate a new `VString` object to hold the token to output for the first column. It then copies the token's value into the `VString` object.
- Calls `setInt()` to output the row ID in the second column.
- Calls `setInt()` again to output the token's position within the input string.

After setting the three column values, the example calls `PartitionWriter.next()` to complete the output row.

TransformFunctionFactory Implementation

The following code shows the factory class.

```
// Provides Vertica with metadata about the StringTokenizer
class TokenFactory : public TransformFunctionFactory {
    // Tell Vertica that StringTokenizer reads 1 int and 1 string,
    // and returns two ints and a string
    virtual void getPrototype(ServerInterface &srvInterface, ColumnTypes
        &argTypes, ColumnTypes &returnType) {
        argTypes.addInt(); // Row number
        argTypes.addVarchar(); // Line of text
        returnType.addVarchar(); // The token
        returnType.addInt(); // The row this token occurred in
        returnType.addInt(); // The position in the row of the token
    }

    // Gives details of the output columns. For integer rows,
    // Tell Vertica the maximum return string length will be, given the input
    // string length. Also names the output columns. This function is only
    // necessary if you have a column that has a variable size (i.e. strings) or
    // have to report their precision.
    virtual void getReturnType(ServerInterface &srvInterface,
        const SizedColumnTypes &input_types,
```

```
        SizedColumnTypes &output_types) {  
  
    // String column's output size will never be more than the size of  
    // the input string  
    int input_len = input_types.getColumnType(1).getStringLength();  
  
    // Describe each column in the output. Adds a column name.  
    output_types.addVarchar(input_len, "token");  
    output_types.addInt("row_id");  
    output_types.addInt("token_position");  
    }  
  
    // Instantiate a StringTokenizer object to process a partition of data  
    virtual TransformFunction *createTransformFunction(ServerInterface  
        &srvInterface) {  
        return vt_createFuncObj(srvInterface.allocator, StringTokenizer);  
    }  
};
```

In this example:

- The UDTF takes an INTEGER and a VARCHAR column as input. To define these input columns, `getPrototype()` calls `addInt()` and `addVarchar()` on the `ColumnTypes` object that represents the input table.
- The UDTF returns a VARCHAR and two INTEGER columns as output. The `getPrototype()` function calls `addVarchar()` once and `addInt()` twice to define the output table.

This example must return the maximum length of the VARCHAR output column. It sets the length to the length input string. This is a safe value, because the output will never be longer than the input string. It also sets the name of the VARCHAR output column to "token".

You can use `getReturnType()` to name the output columns, even if they do not require a precision or output length. The example sets the names of the two INTEGER output columns to "row_id" and "token_postion".

Note: You are not required to supply a name for an output column in this function. However, it is a best practice to do so. If you do not name an output column, `getReturnType()` sets the column name to "". The SQL statements that call your UDTF must provide aliases for any unnamed columns to access them or else they return an error. From a usability standpoint, it is easier for you to supply the column names here once. The alternative is to force all of the users of your function to supply their own column names for each call to the UDTF.

The implementation of the `createTransformFunction()` function in the example is boilerplate code. It just calls the `vt_returnFuncObj` macro with the name of the `TransformFunction` class associated with this factory class. This macro takes care of instantiating a copy of the `TransformFunction` class that Vertica can use to process data.

The RegisterFactory Macro

The final step in creating your UDTF is to call the `RegisterFactory` macro. This macro ensures that your factory class is instantiated when Vertica loads the shared library containing your UDTF. Having your factory class instantiated is the only way that Vertica can find your UDTF and determine what its inputs and outputs are.

The `RegisterFactory` macro just takes the name of your factory class:

```
RegisterFactory(TokenFactory);
```

C++ Example: Multi-Phase Transform Function

The following code fragment is from the `InvertedIndex` UDTF example distributed with the Vertica SDK. It demonstrates subclassing the `MultiPhaseTransformFunctionFactory` including two `TransformFunctionPhase` subclasses that define the two phases in this UDTF.

```
class InvertedIndexFactory : public MultiPhaseTransformFunctionFactory
{
public:
/**
 * Extracts terms from documents.
 */
class ForwardIndexPhase : public TransformFunctionPhase
{
    virtual void getReturnType(ServerInterface &srvInterface,
                               const SizedColumnTypes &inputTypes,
                               SizedColumnTypes &outputTypes)
    {
        // Sanity checks on input we've been given.
        // Expected input: (doc_id INTEGER, text VARCHAR)
        vector<size_t> argCols;
        inputTypes.getArgumentColumns(argCols);
        if (argCols.size() < 2 ||
            !inputTypes.getColumnType(argCols.at(0)).isInt() ||
            !inputTypes.getColumnType(argCols.at(1)).isVarchar())
            vt_report_error(0, "Function only accepts two arguments"
                           "(INTEGER, VARCHAR)");

        // Output of this phase is:
        // (term_freq INTEGER) OVER(PBY term VARCHAR OBY doc_id INTEGER)
        // Number of times term appears within a document.
        outputTypes.addInt("term_freq");
        // Add analytic clause columns: (PARTITION BY term ORDER BY doc_id).
        // The length of any term is at most the size of the entire document.
        outputTypes.addVarcharPartitionColumn(
            inputTypes.getColumnType(argCols.at(1)).getStringLength(),
            "term");
        // Add order column on the basis of the document id's data type.
        outputTypes.addOrderColumn(inputTypes.getColumnType(argCols.at(0)),
                                   "doc_id");
    }
}
```

```
virtual TransformFunction *createTransformFunction(ServerInterface
    &srvInterface)
{ return vt_createFuncObj(srvInterface.allocator, ForwardIndexBuilder); }
};
/**
 * Constructs terms' posting lists.
 */
class InvertedIndexPhase : public TransformFunctionPhase
{
    virtual void getReturnType(ServerInterface &srvInterface,
        const SizedColumnTypes &inputTypes,
        SizedColumnTypes &outputTypes)
    {
        // Sanity checks on input we've been given.
        // Expected input:
        // (term_freq INTEGER) OVER(PBY term VARCHAR OBY doc_id INTEGER)
        vector<size_t> argCols;
        inputTypes.getArgumentColumns(argCols);
        vector<size_t> pByCols;
        inputTypes.getPartitionByColumns(pByCols);
        vector<size_t> oByCols;
        inputTypes.getOrderByColumns(oByCols);
        if (argCols.size() != 1 || pByCols.size() != 1 || oByCols.size() != 1 ||
            !inputTypes.getColumnType(argCols.at(0)).isInt() ||
            !inputTypes.getColumnType(pByCols.at(0)).isVarchar() ||
            !inputTypes.getColumnType(oByCols.at(0)).isInt())
            vt_report_error(0, "Function expects an argument (INTEGER) with "
                "analytic clause OVER(PBY VARCHAR OBY INTEGER)");
        // Output of this phase is:
        // (term VARCHAR, doc_id INTEGER, term_freq INTEGER, corp_freq INTEGER).
        outputTypes.addVarchar(inputTypes.getColumnType(
            pByCols.at(0)).getStringLength(), "term");
        outputTypes.addInt("doc_id");
        // Number of times term appears within the document.
        outputTypes.addInt("term_freq");
        // Number of documents where the term appears in.
        outputTypes.addInt("corp_freq");
    }

    virtual TransformFunction *createTransformFunction(ServerInterface
        &srvInterface)
    { return vt_createFuncObj(srvInterface.allocator, InvertedIndexBuilder); }
};
ForwardIndexPhase fwdIdxPh;
InvertedIndexPhase invIdxPh;
virtual void getPhases(ServerInterface &srvInterface,
    std::vector<TransformFunctionPhase *> &phases)
{
    fwdIdxPh.setPrepass(); // Process documents wherever they're originally stored.
    phases.push_back(&fwdIdxPh);
    phases.push_back(&invIdxPh);
}
virtual void getPrototype(ServerInterface &srvInterface,
    ColumnTypes &argTypes,
    ColumnTypes &returnType)
{
    // Expected input: (doc_id INTEGER, text VARCHAR).
    argTypes.addInt();
    argTypes.addVarchar();
    // Output is: (term VARCHAR, doc_id INTEGER, term_freq INTEGER, corp_freq INTEGER)
```

```
returnType.addVarchar();
returnType.addInt();
returnType.addInt();
returnType.addInt();
}
};
RegisterFactory(InvertedIndexFactory);
```

Most of the code in this example is similar to the code in a `TransformFunctionFactory` class:

- Both `TransformFunctionPhase` subclasses implement the `getReturnType()` function, which describes the output of each stage. This is similar to the `getReturnType()` function from the `TransformFunctionFactory` class. However, this function also lets you control how the data is partitioned and ordered between each phase of your multi-phase UDTF.

The first phase calls `SizedColumnTypes::addVarcharPartitionColumn()` (rather than just `addVarcharColumn()`) to set the phase's output table to be partitioned by the column containing the extracted words. It also calls `SizedColumnTypes::addOrderColumn()` to order the output table by the document ID column. It calls this function instead of one of the data-type-specific functions (such as `addIntOrderColumn()`) so it can pass the data type of the original column through to the output column.

Note: Any order by column or partition by column set by the final phase of the UDTF in its `getReturnType()` function is ignored. Its output is returned to the initiator node rather than partitioned and reordered then sent to another phase.

- The `MultiPhaseTransformFunctionFactory` class implements the `getPrototype()` function, that defines the schemas for the input and output of the multi-phase UDTF. This function is the same as the `TransformFunctionFactory::getPrototype()` function.

The unique function implemented by the `MultiPhaseTransformFunctionFactory` class is `getPhases()`. This function defines the order in which the phases are executed. The fields that represent the phases are pushed into this vector in the order they should execute.

The `MultiPhaseTransformFunctionFactory.getPhases()` function is also where you flag the first phase of the UDTF as operating on data stored locally on the node (called a "pre-pass" phase) rather than on data partitioned across all nodes. Using this option increases the efficiency of your multi-phase UDTF by avoiding having to move significant amounts of data around the Vertica cluster.

Note: Only the first phase of your UDTF can be a pre-pass phase. You cannot have multiple pre-pass phases, and no later phase can be a pre-pass phase.

To mark the first phase as pre-pass, you call the `TransformFunctionPhase::setPrepass()` function of the first phase's `TransformFunctionPhase` instance from within the `getPhase()` function.

Notes

- You need to ensure that the output schema of each phase matches the input schema expected by the next phase. In the example code, each `TransformFunctionPhase::getReturnType()` implementation performs a sanity check on its input and output schemas. Your `TransformFunction` subclasses can also perform these checks in their `processPartition()` function.
- There is no built-in limit on the number of phases that your multi-phase UDTF can have. However, more phases use more resources. When running in fenced mode, Vertica may terminate UDTFs that use too much memory. See [Resource Use for C++ UDxs](#).

Java API

This section provides APIs and examples for the Java API for UDTFs.

For information on setting up a Java development environment and compiling and packaging libraries, see [Developing with the Java SDK](#).

TransformFunction and TransformFunctionFactory Java Interface

This section describes information that is specific to the Java API. See [UDTF Class Overview](#) for general information about implementing the `TransformFunction` and `TransformFunctionFactory` classes.

TransformFunction API

The API provides the following methods for extension by subclasses:

```
void setup(ServerInterface srvInterface, SizedColumnTypes argTypes);  
  
abstract void processPartition(ServerInterface srvInterface,
```

```
PartitionReader input_reader, PartitionWriter input_writer)  
throws UdfException, DestroyInvocation;  
  
void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes);
```

TransformFunctionFactory API

The API provides the following methods for extension by subclasses:

```
abstract TransformFunction createTransformFunction(ServerInterface srvInterface);  
  
abstract void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType);  
  
abstract void getReturnType(ServerInterface srvInterface, SizedColumnTypes argTypes,  
    SizedColumnTypes returnType) throws UdfException;  
  
void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);
```

MultiPhaseTransformFunctionFactory API

The `MultiPhaseTransformFunctionFactory` class extends `TransformFunctionFactory`. The API provides the following additional methods for extension by subclasses:

```
abstract void getPhases(ServerInterface srvInterface,  
    Vector< TransformFunctionPhase > phases);
```

If using this factory you must also extend `TransformFunctionPhase`. See the SDK reference documentation.

Java Example: String Tokenizer

The following example shows a subclass of `TransformFunction` named `StringTokenizer`. It defines a UDTF that reads a table containing an `INTEGER` ID column and a `VARCHAR` column. It breaks the text in the `VARCHAR` column into tokens (individual words). It returns a table containing each token, the row it occurred in, and its position within the string.

You can then use it from SQL statements, for example:

```
=> CREATE TABLE T (url varchar(30), description varchar(2000));  
CREATE TABLE  
=> INSERT INTO T VALUES ('www.amazon.com', 'Online retail merchant and provider of cloud services');  
OUTPUT
```

```
-----
      1
(1 row)
=> INSERT INTO T VALUES ('www.hp.com','Leading provider of computer hardware and imaging solutions');
OUTPUT
-----
      1
(1 row)
=> INSERT INTO T VALUES ('www.vertica.com','World''s fastest analytic database');
OUTPUT
-----
      1
(1 row)
=> COMMIT;
COMMIT

=> -- Invoke the UDT
=> SELECT url, tokenize(description) OVER (partition by url) FROM T;
      url      | words
-----+-----
www.amazon.com | Online
www.amazon.com | retail
www.amazon.com | merchant
www.amazon.com | and
www.amazon.com | provider
www.amazon.com | of
www.amazon.com | c
www.amazon.com | loud
www.amazon.com | services
www.hp.com     | Leading
www.hp.com     | provider
www.hp.com     | of
www.hp.com     | computer
www.hp.com     | hardware
www.hp.com     | and
www.hp.com     | im
www.hp.com     | aging
www.hp.com     | solutions
www.vertica.com | World's
www.vertica.com | fastest
www.vertica.com | analytic
www.vertica.com | database
(22 rows)
```

TransformFunction Implementation

The following code defines the StringTokenizer class.

To make code management simpler, the TransformFunction class is defined as an inner class of the TransformFactoryClass.

```
// You will need to specify the full package when creating functions based on // the classes in your library.
package com.mycompany.example;
// Import the entire Vertica SDK
import com.vertica.sdk.*;
```

```
public class TokenFactory extends TransformFunctionFactory
{
    // Set the number and data types of the columns in the input and output rows.
    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes, ColumnTypes returnType)
    {
        // Define two input columns: an INTEGER and a VARCHAR
        argTypes.addInt(); // Row id
        argTypes.addVarchar(); // Line of text

        // Define the output columns
        returnType.addVarchar(); // The token
        returnType.addInt(); // The row in which this token occurred
        returnType.addInt(); // The position in the row of the token
    }

    // Set the width of any variable-width output columns, and also name
    // them.
    @Override
    public void getReturnType(ServerInterface srvInterface, SizedColumnTypes
        inputTypes, SizedColumnTypes outputTypes)
    {
        // Set the maximum width of the token return column to the width
        // of the input text column and name the output column "Token"
        outputTypes.addVarchar(
            inputTypes.getColumnType(1).getStringLength(), "token");
        // Name the two INTEGER output columns
        outputTypes.addInt("row_id");
        outputTypes.addInt("token_position");
    }

    // Inner class that actually performs work.
    public class TokenizeString extends TransformFunction
    {
        @Override
        public void processPartition(ServerInterface srvInterface,
            PartitionReader inputReader,
            PartitionWriter outputWriter)
            throws UdfException, DestroyInvocation
        {
            try {
                // Loop over all rows passed in in this partition.
                do {
                    // Test if the row ID is null. If so, then do not
                    // process any further. Skip to next row of input.
                    if(inputReader.isLongNull(0)) {
                        srvInterface.log("Skipping row with null id.");
                        continue; // Move on to next row of input
                    }
                    // Get the row ID now that we know it has a value
                    long rowId = inputReader.getLong(0);

                    // Test if the input string is NULL. If so, return NULL
                    // for token and string position.
                    if (inputReader.isStringNull(1)) {
                        outputWriter.setStringNull(0);
                        outputWriter.setLong(1, rowId);
                        outputWriter.setLongNull(2);
                    }
                } while (inputReader.hasNext());
            } catch (UdfException e) {
                throw e;
            }
        }
    }
}
```

```
        outputWriter.next(); // Move to next line of output
    } else {
        // Break string into tokens. Output each word as its own
        // value.
        String[] tokens = inputReader.getString(1).split("\\s+");
        // Output each token on a separate row.
        for (int i = 0; i < tokens.length; i++) {
            outputWriter.getStringWriter(0).copy(tokens[i]);
            outputWriter.setLong(1,rowId);
            outputWriter.setLong(2,i);
            outputWriter.next(); // Advance to next row of output
        }
    }
    // Loop until there are no more input rows in partition.
    } while (inputReader.next());
}
// Prevent exceptions from bubbling back up to server. Uncaught
// exceptions will cause a transaction rollback.
catch (Exception e) {
    // Use more robust error handling in your own
    // UDTFs. This example just sends a message to the log.
    srvInterface.log("Exception: " + e.getClass().getSimpleName()
        + "Message: " + e.getMessage());
}
}
}

@Override
public TransformFunction createTransformFunction(ServerInterface srvInterface)
{ return new TokenizeString(); }
```

Java Example: Multi-Phase Transform Function

The following code is excerpted from the InvertedIndexFactory SDK example. You can find the complete code in `/opt/vertica/sdk/examples/JavaUDx/TransformFunctions`.

```
public class InvertedIndexFactory extends MultiPhaseTransformFunctionFactory {

    public class ForwardIndexPhase extends TransformFunctionPhase {
        // ...
    }

    public class InvertedIndexPhase extends TransformFunctionPhase {

        @Override
        public TransformFunction
        createTransformFunction(ServerInterface srvInterface) {
            return new InvertedIndexBuilder();
        }

        @Override
        public void getReturnType(ServerInterface srvInterface,
            SizedColumnTypes inputTypes,
```

```
        SizedColumnTypes outputTypes) {
// Sanity checks on input we've been given.
// Expected input:
// (term_freq INTEGER) OVER(PBY term VARCHAR OBY doc_id INTEGER)
ArrayList<Integer> argCols = new ArrayList<Integer>();
inputTypes.getArgumentColumns(argCols);

ArrayList<Integer> pByCols = new ArrayList<Integer>();
inputTypes.getPartitionByColumns(pByCols);

ArrayList<Integer> oByCols = new ArrayList<Integer>();
inputTypes.getOrderByColumns(oByCols);

if (argCols.size() != 1 || pByCols.size() != 1
    || oByCols.size() != 1
    || !inputTypes.getColumnType(argCols.get(0)).isInt()
    || !inputTypes.getColumnType(pByCols.get(0)).isVarchar()
    || !inputTypes.getColumnType(oByCols.get(0)).isInt()) {
    throw new UdfException(
        0,
        "Function expects an argument (INTEGER) with "
        + "analytic clause OVER(PBY VARCHAR OBY INTEGER)");
}

// Output of this phase is:
// (term VARCHAR, doc_id INTEGER, term_freq INTEGER, corp_freq
// INTEGER).
outputTypes.addVarchar(inputTypes.getColumnType(pByCols.get(0))
    .getStringLength(), "term");
outputTypes.addInt("doc_id");

// Number of times term appears within the document.
outputTypes.addInt("term_freq");

// Number of documents where the term appears in.
outputTypes.addInt("corp_freq");

}
}

@Override
public void getPhases(ServerInterface srvInterface,
    Vector<TransformFunctionPhase> phases) {
    ForwardIndexPhase fwdIdxPh;
    InvertedIndexPhase invIdxPh;

    fwdIdxPh = new ForwardIndexPhase();
    invIdxPh = new InvertedIndexPhase();

    fwdIdxPh.setPrepass();
    phases.add(fwdIdxPh);
    phases.add(invIdxPh);
}

@Override
public void getPrototype(ServerInterface srvInterface,
    ColumnTypes argTypes, ColumnTypes returnTypes) {
// Expected input: (doc_id INTEGER, text VARCHAR).
argTypes.addInt();
```

```
argTypes.addVarchar();

// Output is: (term VARCHAR, doc_id INTEGER, term_freq INTEGER,
// corp_freq INTEGER)
returnTypes.addVarchar();
returnTypes.addInt();
returnTypes.addInt();
returnTypes.addInt();
}
}
```

Most of the code in this example is similar to the code in a `TransformFunctionFactory` class. The `getReturnType` method is similar, but in a `TransformFunctionPhase` it can also partition and order the data. The partitions and order are used by the next phase, but they are ignored after the final phase.

The `MultiPhaseTransformFunctionFactory` class adds the `getPhases` method. This method defines the order in which the phases are executed. The fields that represent the phases are pushed into this vector in the order they should execute. You should also indicate the first phase by calling `setPrepass()` on it. Doing so can lead to better performance.

There is no built-in limit on the number of phases that your multi-phase UDTF can have. However, more phases use more resources. Vertica might terminate UDTFs that use too much memory.

R API

This section provides APIs and examples for the R API for UDTFs.

For information on setting up an R development environment and compiling and packaging libraries, see [Developing with the R SDK](#).

Transform Function and Transform Function Factory R Interface

This section describes information that is specific to the R API. See [R SDK API Documentation](#) for general information about implementing transform functions.

Transform Function API

The API provides the following framework for extension by R function:

```
FunctionName <- function(input.data.frame, parameters.data.frame) {
  # Computations
}
```

```
# The function must return a data frame.  
return(output.data.frame)  
}
```

Transform Function Factory API

The API provides the following framework for extension by R function:

```
FunctionNameFactory <- function() {  
  list(name = FunctionName,  
        udxtype = c("transform"),  
        intype = c("int"),  
        outtype = c("int"))  
}
```

R Example: Log Tokenizer

The `LogTokenizer` transform function reads a varchar from a table, a log message. It then tokenizes each of the log messages, returning each of the tokens.

You can find more UDX examples in the VerticaGithub repository, <https://github.com/vertica/UDx-Examples>.

Load the Function and Library

Create the library and the function.

```
=> CREATE OR REPLACE LIBRARY rLib AS 'log_tokenizer.R' LANGUAGE 'R';  
CREATE LIBRARY  
=> CREATE OR REPLACE TRANSFORM FUNCTION LogTokenizer AS LANGUAGE 'R' NAME 'LogTokenizerFactory'  
LIBRARY rLib FENCED;  
CREATE FUNCTION
```

Querying Data with the Function

The following query shows how you can run a query with the UDTF.

```
=> SELECT machine,  
        LogTokenizer(error_log USING PARAMETERS spliton = ' ') OVER(PARTITION BY machine)  
FROM error_logs;  
machine | Token  
-----+-----  
node001 | ERROR  
node001 | 345  
node001 | -
```

```
node001 | Broken
node001 | pipe
node001 | WARN
node001 | -
node001 | Nearly
node001 | filled
node001 | disk
node002 | ERROR
node002 | 111
node002 | -
node002 | Flooded
node002 | roads
node003 | ERROR
node003 | 222
node003 | -
node003 | Plain
node003 | old
node003 | broken
(21 rows)
```

UDTF R Code

```
LogTokenizer <- function(input.data.frame, parameters.data.frame) {
  # Take the spliton parameter passed by the user and assign it to a variable
  # in the function so we can use that as our tokenizer.
  if ( is.null(parameters.data.frame[["spliton"]]) ) {
    stop("NULL value for spliton! Token cannot be NULL.")
  } else {
    split.on <- as.character(parameters.data.frame[["spliton"]])
  }
  # Tokenize the string.
  tokens <- vector(length=0)
  for ( string in input.data.frame[, 1] ) {
    tokenized.string <- strsplit(string, split.on)
    for ( token in tokenized.string ) {
      tokens <- append(tokens, token)
    }
  }
  final.output <- data.frame(tokens)
  return(final.output)
}

LogTokenizerFactory <- function() {
  list(name = LogTokenizer,
       udxtype = c("transform"),
       intype = c("varchar"),
       outtype = c("varchar"),
       outtypecallback=LogTokenizerReturn,
       parametertypecallback=LogTokenizerParameters)
}

LogTokenizerParameters <- function() {
  parameters <- list(datatype = c("varchar"),
                    length = c("NA"),
                    scale = c("NA"),
```

```
        name = c("spliton"))
    return(parameters)
}

LogTokenizerReturn <- function(arg.data.frame, parm.data.frame) {
  output.return.type <- data.frame(datatype = rep(NA,1),
    length = rep(NA,1),
    scale = rep(NA,1),
    name = rep(NA,1))
  output.return.type$datatype <- c("varchar")
  output.return.type$name <- c("Token")
  return(output.return.type)
}
```

Load (UDLs)

The [COPY statement](#) offers extensive options and settings to control how to load data. However, you may find that these options do not suit the type of data load that you want to perform. The User-Defined Load (UDL) feature lets you develop one or more functions that change how the COPY statement operates. You can create custom libraries using the Vertica SDK to handle various steps in the loading process. The SDK provides APIs for C++ and Java.

You use three types of UDL functions during development, one for each stage of the data-load process:

- [User-Defined Source](#) (UDSource): Controls how the COPY statement obtains the data it loads into the database. For example, COPY might obtain data by fetching it through HTTP or cURL. Up to one UDSource reads data from a file or input stream. Your UDSource can read from more than one source, but COPY invokes only one UDSource.
- [User-Defined Filter](#) (UDFilter): Preprocesses the data. For example, a filter might unzip a file or convert UTF-16 to UTF-8. You can chain multiple User-Defined Filters together, for example unzipping and then converting.
- [User-Defined Parser](#) (UDParser): Up to one parser parses the data into tuples that are ready to be inserted into a table. For example, a parser could extract data from an XML-like format. You can optionally define a User-Defined Chunker (UDChunker, C++ only), to have the parser perform parallel parsing.

After the final step, COPY inserts the data into a table, or rejects it, if it is not in the correct format.

Important: You cannot use fenced-mode UDLs developed in different programming languages together in the same COPY statement. For example, you cannot have a User-Defined Source written in C++ send data to a User-Defined Filter developed in Java. Attempting to do so results in a "Failure in UDx RPC call" error message.

User-Defined Source

A User-Defined Source allows you to process a source of data using a method that is not built into Vertica. For example, you can write a User-Defined Source to access the data from an HTTP source using cURL. While you can use only a single User-Defined Source in a [COPY](#) statement, that source function can pull data from multiple sources.

The `UDSource` class is responsible for acquiring data from an external source. It reads data from an input stream and produces an output stream to be filtered and parsed. If you implement a `UDSource`, you must also implement a corresponding `SourceFactory`.

See [UDSource Class](#) and [SourceFactory Class](#) for API details.

UDSource Class

You can subclass the `UDSource` class when you need to load data from a source type that `COPY` does not already support.

Each instance of your `UDSource` subclass reads from a single data source. Examples of a single data source are a single file or the results of a single function call to a RESTful web application.

UDSource Methods

The `UDSource` class defines the following methods. Your subclass must override `process()`. You can optionally override the others.

For the signatures of these methods and language-specific information, see [Source Classes \(C++\)](#) and [Source Classes \(Java\)](#).

Setting Up

`COPY` calls `setup()` before the first time it calls `process()`. Use `setup()` to perform any necessary setup steps to access the data source. This method establishes network connections, opens files, and similar tasks that need to be performed before the `UDSource` can read data from the data source. Your object might be destroyed and re-created during use, so make sure that your object is restartable.

Processing a Source

`COPY` calls `process()` repeatedly during query execution to read data and write it to the `DataBuffer` passed as a parameter. This buffer is then passed to the first filter.

If the source runs out of input, or fills the output buffer, it must return the value `StreamState.OUTPUT_NEEDED`. When Vertica gets this return value, it will call the method again. This second call occurs after the output buffer has been processed by the next stage in

the data-load process. Returning `StreamState.DONE` indicates that all of the data from the source has been read.

The user can cancel the load operation, which aborts reading.

Tearing Down

`COPY` calls `destroy()` after the last time that `process()` is called. This method frees any resources reserved by the `setup()` or `process()` methods, such as file handles or network connections that the `setup()` method allocated.

Accessors

A source can define two accessors, `getSize()` and `getUri()`.

`COPY` might call `getSize()` to estimate the number of bytes of data to be read before calling `process()`. This value is an estimate only and is used to indicate the file size in the `LOAD_STREAMS` table. Because Vertica can call this method before calling `setup()`, `getSize()` must not rely on any resources allocated by `setup()`.

This method should not leave any resources open. For example, do not save any file handles opened by `getSize()` for use by the `process()` method. Doing so can exhaust the available resources, because Vertica calls `getSize()` on all instances of your `UDSource` subclass before any data is loaded. If many data sources are being opened, these open file handles could use up the system's supply of file handles. Thus, none would remain available to perform the actual data load.

Vertica calls `getUri()` during execution to update status information about which resources are currently being loaded. It returns the URI of the data source being read by this `UDSource`.

SourceFactory Class

If you write a source, you must also write a source factory. Your subclass of the `SourceFactory` class is responsible for:

- Performing the initial validation of the parameters passed to your `UDSource`.
- Setting up any data structures your `UDSource` instances need to perform their work. This information can include recording which nodes will read which data source.

- Creating one instance of your `UDSource` subclass for each data source (or portion thereof) that your function reads on each host.

The simplest source factory creates one `UDSource` instance per data source per executor node. You can also use multiple concurrent `UDSource` instances on each node. This behavior is called *concurrent load*. To support both options, `SourceFactory` has two versions of the method that creates the sources. You must implement exactly one of them.

Source factories are singletons. Your subclass must be stateless, with no fields containing data. The subclass also must not modify any global variables.

SourceFactory Methods

The `SourceFactory` class defines several methods. Your class must override `prepareUDSources()`; it may override the other methods.

Setting Up

Vertica calls `plan()` once on the initiator node to perform the following tasks:

- Check the parameters the user supplied to the function call in the `COPY` statement and provide error messages if there are any issues. You can read the parameters by getting a `ParamReader` object from the instance of `ServerInterface` passed into the `plan()` method.
- Decide which hosts in the cluster will read the data source. How you divide up the work depends on the source your function is reading. Some sources can be split across many hosts, such as a source that reads data from many URLs. Others, such as an individual local file on a host's filesystem, can be read only by a single specified host.

You store the list of hosts to read the data source by calling the `setTargetNodes()` method on the `NodeSpecifyingPlanContext` object. This object is passed into your `plan()` method.

- Store any information that the individual hosts need to process the data sources in the `NodeSpecifyingPlanContext` instance passed to the `plan()` method. For example, you could store assignments that tell each host which data sources to process. The `plan()` method runs only on the initiator node, and the `prepareUDSources()` method runs on each host reading from a data source. Therefore, this object is the only means of communication between them.

You store data in the `NodeSpecifyingPlanContext` by getting a `ParamWriter` object from the `getWriter()` method. You then write parameters by calling methods on the `ParamWriter` such as `setString()`.

Note: `ParamWriter` offers the ability to store only simple data types. For complex types, you must serialize the data in some manner and store it as a string or long string.

Creating Sources

Vertica calls `prepareUDSources()` on all hosts that the `plan()` method selected to load data. This call instantiates and returns a list of `UDSource` subclass instances. If you are not using concurrent load, return one `UDSource` for each of the sources that the host is assigned to process. If you are using concurrent load, use the version of the method that takes an `ExecutorPlanContext` as a parameter, and return as many sources as you can use. Your factory must implement exactly one of these methods.

Note: In the C++ API, the function that supports concurrent load is named `prepareUDSourcesExecutor()`. In the Java API the class provides two overloads of `prepareUDSources()`.

For concurrent load, you can find out how many threads are available on the node to run `UDSource` instances by calling `getLoadConcurrency()` on the `ExecutorPlanContext` that is passed in.

Defining Parameters

Implement `getParameterTypes()` to define the names and types of parameters that your source uses. Vertica uses this information to warn callers about unknown or missing parameters. Vertica ignores unknown parameters and uses default values for missing parameters. While you should define the types and parameters for your function, you are not required to override this method.

Requesting Threads for Concurrent Load

When a source factory creates sources on an executor node, by default, it creates one thread per source. If your sources can use multiple threads, implement `getDesiredThreads()`. Vertica calls this method before it calls `prepareUDSources()`, so you can also use it to decide how many sources to create. Return the number of threads your factory can use for sources. The maximum number of available threads is passed in, so you can take that into account. The value your method returns is a hint, not a guarantee; each executor node determines the number of threads to allocate. The `FilePortionSourceFactory` example implements this method; see [Source Example: Concurrent Load](#).

You can allow your source to have control over parallelism, meaning that it can divide a single input into multiple load streams, by implementing `isSourceApportionable()`. Returning `true` from this method does not guarantee that the source *will* apportion the load. However, returning `false` indicates that it will not try to do so. See [Apportioned Load](#) for more information.

Often, a `SourceFactory` that implements `getDesiredThreads()` also uses apportioned load. However, using apportioned load is not a requirement. A source reading from Kafka streams, for example, could use multiple threads without `ssapportioning`.

User-Defined Filter

User-Defined Filter functions allow you to manipulate data obtained from a source in various ways.

For example, a filter could:

- Process a compressed file in a compression format not natively supported by Vertica.
- Take UTF-16-encoded data and transcode it to UTF-8 encoding.
- Perform search-and-replace operations on data before it is loaded into Vertica.

You can also process data through multiple filters before it is loaded into Vertica. For instance, you could unzip a file compressed with GZip, convert the content from UTF-16 to UTF-8, and finally search and replace certain text strings.

If you implement a `UDFilter`, you must also implement a corresponding `FilterFactory`.

See [UDFilter Class](#) and [FilterFactory Class](#) for API details.

UDFilter Class

The `UDFilter` class is responsible for reading raw input data from a source and preparing it to be loaded into Vertica or processed by a parser. This preparation may involve decompression, re-encoding, or any other sort of binary manipulation.

A `UDFilter` is instantiated by a corresponding `FilterFactory` on each host in the Vertica cluster that is performing filtering for the data source.

UDFilter Methods

The `UDFilter` class defines the following methods. Your subclass must override `process()`. You can optionally override the other methods.

For the signatures of these methods and language-specific information, see [Filter Classes \(C++\)](#) and [Filter Classes \(Java\)](#).

Setting Up

`COPY` calls `setup()` before the first time it calls `process()`. Use `setup()` to perform any necessary setup steps that your filter needs to operate, such as initializing data structures to be used during filtering. Your object might be destroyed and re-created during use, so make sure that your object is restartable.

Filtering Data

`COPY` calls `process()` repeatedly during query execution to filter data. The method receives two instances of the `DataBuffer` class among its parameters, an input and an output buffer. Your implementation should read from the input buffer, manipulate it in some manner (such as decompressing it), and write the result to the output. A one-to-one correlation between the number of bytes your implementation reads and the number it writes might not exist. The `process()` method should process data until it either runs out of data to read or runs out of space in the output buffer. When one of these conditions occurs, your method should return one of the following values defined by `StreamState`:

- `OUTPUT_NEEDED` if the filter needs more room in its output buffer.
- `INPUT_NEEDED` if the filter has run out of input data (but the data source has not yet been fully processed).
- `DONE` if the filter has processed all of the data in the data source.
- `KEEP_GOING` if the filter cannot proceed for an extended period of time. The method will be called again, so do not block indefinitely. If you do, then you prevent your user from canceling the query.

Before returning, your `process()` method must set the `offset` property in each `DataBuffer`. In the input buffer, set it to the number of bytes that the method successfully read. In the output buffer, set it to the number of bytes the method wrote. Setting these properties allows the next call to `process()` to resume reading and writing data at the correct points in the buffers.

Your `process()` method also needs to check the `InputState` object passed to it to determine if there is more data in the data source. When this object is equal to `END_OF_FILE`, then the data remaining in the input data is the last data in the data source. Once it has processed all of the remaining data, `process()` must return `DONE`.

Tearing Down

`COPY` calls `destroy()` after the last time it calls `process()`. This method frees any resources reserved by the `setup()` or `process()` methods. Vertica calls this method after the `process()` method indicates it has finished filtering all of the data in the data stream.

If there are still data sources that have not yet been processed, Vertica may later call `setup()` on the object again. On subsequent calls Vertica directs the method to filter the data in a new data stream. Therefore, your `destroy()` method should leave an object of your `UDFilter` subclass in a state where the `setup()` method can prepare it to be reused.

FilterFactory Class

If you write a filter, you must also write a filter factory to produce filter instances. To do so, subclass the `FilterFactory` class.

Your subclass performs the initial validation and planning of the function execution and instantiates `UDFilter` objects on each host that will be filtering data.

Filter factories are singletons. Your subclass must be stateless, with no fields containing data. The subclass also must not modify any global variables.

FilterFactory Methods

The `FilterFactory` class defines the following methods. Your subclass must override the `prepare()` method. It may override the other methods.

Setting Up

Vertica calls `plan()` once on the initiator node, to perform the following tasks:

- Check any parameters that have been passed from the function call in the COPY statement and error messages if there are any issues. You read the parameters by getting a `ParamReader` object from the instance of `ServerInterface` passed into your `plan()` method.
- Store any information that the individual hosts need in order to filter data in the `PlanContext` instance passed as a parameter. For example, you could store details of the input format that the filter will read and output the format that the filter should produce. The `plan()` method runs only on the initiator node, and the `prepare()` method runs on each host reading from a data source. Therefore, this object is the only means of communication between them.

You store data in the `PlanContext` by getting a `ParamWriter` object from the `getWriter()` method. You then write parameters by calling methods on the `ParamWriter` such as `setString`.

Note: `ParamWriter` offers only the ability to store simple data types. For complex types, you need to serialize the data in some manner and store it as a string or long string.

Creating Filters

Vertica calls `prepare()` to create and initialize your filter. It calls this method once on each node that will perform filtering. Vertica automatically selects the best nodes to complete the work based on available resources. You cannot specify the nodes on which the work is done.

Defining Parameters

Implement `getParameterTypes()` to define the names and types of parameters that your filter uses. Vertica uses this information to warn callers about unknown or missing parameters. Vertica ignores unknown parameters and uses default values for missing parameters. While you should define the types and parameters for your function, you are not required to override this method.

User-Defined Parser

A parser takes a stream of bytes and passes a corresponding sequence of tuples to the Vertica load process. You can use User-Defined Parser functions to parse:

- Data in formats not understood by the Vertica built-in parser.
- Data that requires more specific control than the built-in parser supplies.

For example, you could load a CSV file using a specific CSV library. See the Vertica SDK for two CSV examples.

COPY supports a single User-Defined Parser that you can use with a `UDSource` and zero or more instances of `UDFilter`.

Sometimes you can improve the performance of your parser by adding a *chunker*. A *chunker* divides up the input and uses multiple threads to parse it. See [Cooperative Parse](#). Chunkers are available only in the C++ API.

Under special circumstances you can further improve performance by using *apportioned load*, an approach where multiple Vertica nodes parse the input. See [Apportioned Load](#).

If you implement a `UDParser`, you must also implement a corresponding `ParserFactory`.

See [UDParser Class](#) and [ParserFactory Class](#) for API details.

UDParser Class

You can subclass the `UDParser` class when you need to parse data that is in a format that the COPY statement's native parser cannot handle.

During parser execution, Vertica always calls three methods: `setup()`, `process()`, and `destroy()`. It might also call `getRejectedRecord()`.

UDParser Methods

The `UDParser` class defines the following methods. Your subclass must override the `process()` and `getRejectedRecord()` methods. You can optionally override the other methods.

For the signatures of these methods and language-specific information, see [Parser Classes \(C++\)](#) and [Parser Classes \(Java\)](#).

Note: The `UDParser` class performs important initialization required by all subclasses, including initializing the `StreamWriter` object used by the parser. Therefore, your constructor must call `super()`.

Setting Up

`COPY` calls `setup()` before the first time it calls `process()`. Use `setup()` to perform any initial setup tasks that your parser needs to parse data. This setup includes retrieving parameters from the class context structure or initializing data structures for use during filtering. Vertica calls this method before calling the `process()` method for the first time. Your object might be destroyed and re-created during use, so make sure that your object is restartable.

Parsing

`COPY` calls `process()` repeatedly during query execution. Vertica passes this method a buffer of data to parse into columns and rows and one of the following input states defined by `InputState`:

- `OK`: currently at the start of or in the middle of a stream
- `END_OF_FILE`: no further data is available.
- `END_OF_CHUNK`: the current data ends on a record boundary and the parser should consume all of it before returning. This input state only occurs when using a chunker.

- **START_OF_PORTION**: the input does not start at the beginning of a source. The parser should find the first end-of-record mark. This input state only occurs when using apportioned load. You can use the `getPortion()` method to access the offset and size of the portion.
- **END_OF_PORTION**: the source has reached the end of its portion. The parser should finish processing the last record it started and advance no further. This input state only occurs when using apportioned load.

The parser must reject any data that it cannot parse, so that Vertica can report the rejection and write the rejected data to files.

The `process()` method must parse as much data as it can from the input buffer. The buffer might not end on a row boundary. Therefore, it might have to stop parsing in the middle of a row of input and ask for more data. The input can contain null bytes, if the source file contains them, and is *not* automatically null-terminated.

A parser has an associated `StreamWriter` object, which performs the actual writing of the data. When your parser extracts a column value, it uses one of the type-specific methods on `StreamWriter` to write that value to the output stream. See [Writing Data](#) for more information about these methods.

A single call to `process()` might write several rows of data. When your parser finishes processing a row of data, it must call `next()` on its `StreamWriter` to advance the output stream to a new row. (Usually a parser finishes processing a row because it encounters an end-of-row marker.)

When your `process()` method reaches the end of the buffer, it tells Vertica its current state by returning one of the following values defined by `StreamState`:

- **INPUT_NEEDED**: the parser has reached the end of the buffer and needs more data to parse.
- **DONE**: the parser has reached the end of the input data stream.
- **REJECT**: the parser has rejected the last row of data it read (see [Rejecting Rows](#)).

Tearing Down

`COPY` calls `destroy()` after the last time that `process()` is called. It frees any resources reserved by the `setup()` or `process()` method.

Vertica calls this method after the `process()` method indicates it has completed parsing the data source. However, sometimes data sources that have not yet been processed might remain. In such cases, Vertica might later call `setup()` on the object again and have it parse the data in a new data stream. Therefore, write your `destroy()` method so that it leaves an instance of your `UDParser` subclass in a state where `setup()` can be safely called again.

Reporting Rejections

If `process()` rejects a row, Vertica calls `getRejectedRecord()` to report it. Usually, this method returns an instance of the `RejectedRecord` class with details of the rejected row.

Writing Data

A parser has an associated `StreamWriter` object, which you access by calling `getStreamWriter()`. In your `process()` implementation, use the `setType()` methods on the `StreamWriter` object to write values in a row to specific column indexes. Verify that the data types you write match the data types expected by the schema.

The following example shows how you can write a value of type `long` to the fourth column (index 3) in the current row:

```
StreamWriter writer = getStreamWriter();  
...  
writer.setLongValue(3, 98.6);
```

`StreamWriter` provides methods for all the basic types, such as `setBooleanValue()`, `setStringValue()`, and so on. See the API documentation for a complete list of `StreamWriter` methods, including options that take primitive types or explicitly set entries to null.

The Java API supports additional options for writing data. See [Parser Classes](#).

Rejecting Rows

If your parser finds data it cannot parse, it should reject the row by:

1. Saving details about the rejected row data and the reason for the rejection. These pieces of information can be directly stored in a `RejectedRecord` object, or in fields on your `UDParser` subclass, until they are needed.

2. Updating the row's position in the input buffer by updating `input.offset` so it can resume parsing with the next row.
3. Signaling that it has rejected a row by returning with the value `StreamState.REJECT`.
4. Returning an instance of the `RejectedRecord` class with the details about the rejected row.

Breaking Up Large Loads

Vertica provides two ways to break up large loads. [Apportioned Load](#) allows you to distribute a load among several database nodes. [Cooperative Parse](#) (C++ only) allows you to distribute a load among several threads on one node.

UDChunker Class

You can subclass the `UDChunker` class to allow your parser to support [Cooperative Parse](#). This class is available only in the C++ API.

Fundamentally, a `UDChunker` is a very simplistic parser. Like `UDParser`, it has the following three methods: `setup()`, `process()`, and `destroy()`. You must override `process()`; you may override the others. This class has one additional method, `alignPortion()`, which you must implement if you want to enable [Apportioned Load](#) for your `UDChunker`.

For the signatures of these methods, see [Parser Classes](#).

Setting Up and Tearing Down

As with `UDParser`, you can define initialization and cleanup code for your chunker. Vertica calls `setup()` before the first call to `process()` and `destroy()` after the last call to `process()`. Your object might be reused amongst multiple load sources, so make sure that `setup()` completely initializes all fields.

Chunking

Vertica calls `process()` to divide an input into chunks that can be parsed independently. The method takes an input buffer and an indicator of the input state:

- `OK`: the input buffer begins at the start of or in the middle of a stream.
- `END_OF_FILE`: no further data is available.

- `END_OF_PORTION`: the source has reached the end of its portion. This state occurs only when using apportioned load.

If the input state is `END_OF_FILE`, the chunker should set the `input.offset` marker to `input.size` and return `DONE`. Returning `INPUT_NEEDED` is an error.

If the input state is `OK`, the chunker should read data from the input buffer and find record boundaries. If it finds the end of at least one record, it should align the `input.offset` marker with the byte after the end of the last record in the buffer and return `CHUNK_ALIGNED`. For example, if the input is "abc~def" and "~" is a record terminator, this method should set `input.offset` to 4, the position of "d". If `process()` reaches the end of the input without finding a record boundary, it should return `INPUT_NEEDED`.

You can divide the input into smaller chunks, but consuming all available records in the input can have better performance. For example, a chunker could scan backwards from the end of the input to find a record terminator, which might be the last of many records in the input, and return it all as one chunk without scanning through the rest of the input.

If the input state is `END_OF_PORTION`, the chunker should behave as it does for an input state of `OK`, except that it should also set a flag. When called again, it should find the first record in the next portion and align the chunk to that record.

The input data can contain null bytes, if the source file contains them. The input argument is not automatically null-terminated.

The `process()` method must not block indefinitely. If this method cannot proceed for an extended period of time, it should return `KEEP_GOING`. Failing to return `KEEP_GOING` has several consequences, such as preventing your user from being able to cancel the query.

See [Chunker Example: Delimited Parser and Chunker](#) for an example of the `process()` method using chunking.

Aligning Portions

If your chunker supports apportioned load, implement the `alignPortion()` method. Vertica calls this method one or more times, before calling `process()`, to align the input offset with the beginning of the first complete chunk in the portion. The method takes an input buffer and an indicator of the input state:

- `START_OF_PORTION`: the beginning of the buffer corresponds to the start of the portion. You can use the `getPortion()` method to access the offset and size of the portion.
- `OK`: the input buffer is in the middle of a portion.

- `END_OF_PORTION`: the end of the buffer corresponds to the end of the portion or beyond the end of a portion.
- `END_OF_FILE`: no further data is available.

The method should scan from the beginning of the buffer to the start of the first complete record. It should set `input.offset` to this position and return one of the following values:

- `DONE`, if it found a chunk. `input.offset` is the first byte of the chunk.
- `INPUT_NEEDED`, if the input buffer does not contain the start of any chunk. It is an error to return this from an input state of `END_OF_FILE`.
- `REJECT`, if the portion (not buffer) does not contain the start of any chunk.

ParserFactory Class

If you write a parser, you must also write a factory to produce parser instances. To do so, subclass the `ParserFactory` class.

Parser factories are singletons. Your subclass must be stateless, with no fields containing data. Your subclass also must not modify any global variables.

The `ParserFactory` class defines the following methods. Your subclass must override the `prepare()` method. It may override the other methods.

Setting Up

Vertica calls `plan()` once on the initiator node to perform the following tasks:

- Check any parameters that have been passed from the function call in the `COPY` statement and error messages if there are any issues. You read the parameters by getting a `ParamReader` object from the instance of `ServerInterface` passed into your `plan()` method.
- Store any information that the individual hosts need in order to parse the data. For example, you could store parameters in the `PlanContext` instance passed in through the `planCtxt` parameter. The `plan()` method runs only on the initiator node, and the `prepareUDSources()` method runs on each host reading from a data source. Therefore, this object is the only means of communication between them.

You store data in the `PlanContext` by getting a `ParamWriter` object from the `getWriter()` method. You then write parameters by calling methods on the `ParamWriter` such as `setString`.

Note: `ParamWriter` offers only the ability to store simple data types. For complex types, you need to serialize the data in some manner and store it as a string or long string.

Creating Parsers

Vertica calls `prepare()` on each node to create and initialize your parser, using data stored by the `plan()` method.

Defining Parameters

Implement `getParameterTypes()` to define the names and types of parameters that your parser uses. Vertica uses this information to warn callers about unknown or missing parameters. Vertica ignores unknown parameters and uses default values for missing parameters. While you should define the types and parameters for your function, you are not required to override this method.

Defining Parser Outputs

Implement `getParserReturnType()` to define the data types of the table columns that the parser outputs. If applicable, `getParserReturnType()` also defines the size, precision, or scale of the data types. Usually, this method reads data types of the output table from the `argType` and `perColumnParamReader` arguments and verifies that it can output the appropriate data types. If `getParserReturnType()` is prepared to output the data types, it calls methods on the `SizedColumnTypes` object passed in the `returnType` argument. In addition to the data type of the output column, your method should also specify any additional information about the column's data type:

- For binary and string data types (such as `CHAR`, `VARCHAR`, and `LONG VARBINARY`), specify its maximum length.
- For `NUMERIC` types, specify its precision and scale.
- For `Time/Timestamp` types (with or without time zone), specify its precision (-1 means

unspecified).

- For all other types, no length or precision specification is required.

Supporting Cooperative Parse

To support [Cooperative Parse](#), implement `prepareChunker()` and return an instance of your `UDChunker` subclass. If `isChunkerApportionable()` returns `true`, then it is an error for this method to return `null`.

Cooperative parse is currently supported only in the C++ API.

Supporting Apportioned Load

To support [Apportioned Load](#), your parser, chunker, or both must support apportioning. To indicate that the parser can apportion a load, implement `isParserApportionable()` and return `true`. To indicate that the chunker can apportion a load, implement `isChunkerApportionable()` and return `true`.

The `isChunkerApportionable()` method takes a `ServerInterface` as an argument, so you have access to the parameters supplied in the `COPY` statement. You might need this information if the user can specify a record delimiter, for example. Return `true` from this method if and only if the factory can create a chunker for this input.

Load Parallelism

Vertica can divide the work of loading data, taking advantage of parallelism to speed up the operation. Vertica supports several types of parallelism:

- **Distributed load:** Vertica distributes files in a multi-file load to several nodes to load in parallel, instead of loading all of them on a single node. Vertica manages distributed load; you do not need to do anything special in your UDL.
- **Cooperative parse:** A source being loaded on a single node uses multi-threading to parallelize the parse. Cooperative parse divides a load at execution time, based on how threads are scheduled. You must enable cooperative parse in your parser. See [Cooperative Parse](#).
- **Apportioned load:** Vertica; divides a single large file or other single source into segments, which it assigns to several nodes to load in parallel. Apportioned load divides the load at

planning time, based on available nodes and cores on each node. You must enable apporioned load in your source and parser. See [Apporioned Load](#).

You can support both cooperative parse and apporioned load in the same UDL. Vertica decides which to use for each load operation and might use both. See [Combining Cooperative Parse and Apporioned Load](#).

Cooperative Parse

By default, Vertica parses a data source in a single thread on one database node. You can optionally use *cooperative parse* to parse a source using multiple threads on a node. More specifically, data from a source passes through a *chunker* that groups blocks from the source stream into logical units. These chunks can be parsed in parallel. The chunker divides the input into pieces that can be individually parsed, and the parser then parses them concurrently. Cooperative parse is available only for unfenced UDxs. (See [Fenced Mode](#).)

To use cooperative parse, a chunker must be able to locate end-of-record markers in the input. Locating these markers might not be possible in all input formats.

Chunkers are created by parser factories. At load time, Vertica first calls the UDChunker to divide the input into chunks and then calls the UDParser to parse each chunk.

You can use cooperative parse and apporioned load independently or together. See [Combining Cooperative Parse and Apporioned Load](#).

How Vertica Divides a Load

When Vertica receives data from a source, it calls the chunker's `process()` method repeatedly. A chunker is, essentially, a lightweight parser; instead of parsing, the `process()` method divides the input into chunks.

After the chunker has finished dividing the input into chunks, Vertica sends those chunks to as many parsers as are available, calling the `process()` method on the parser.

Implementing Cooperative Parse

To implement cooperative parse, perform the following actions:

- Subclass UDChunker and implement `process()`.
- In your ParserFactory, implement `prepareChunker()` to return a UDChunker.

See [Chunker Example: Delimited Parser and Chunker](#) for a UDChunker that also supports apporportioned load.

Apporportioned Load

A parser can use more than one database node to load a single input source in parallel. This approach is referred to as *apporportioned load*. Some of the parsers built into Vertica support apporportioned load.

Apporportioned load, like cooperative parse, requires an input that can be divided at record boundaries. The difference is that cooperative parse does a sequential scan to find record boundaries, while apporportioned load first jumps (seeks) to a given position and then scans. Some formats, like generic XML, do not support seeking.

To use apporportioned load, you must ensure that the source is reachable by all participating database nodes. You typically use apporportioned load with distributed file systems.

It is possible for a parser to not support apporportioned load directly but to have a chunker that supports apporportioning.

You can use apporportioned load and cooperative parse independently or together. See [Combining Cooperative Parse and Apporportioned Load](#).

How Vertica Apporportions a Load

If both the parser and its source support apporportioning, then you can specify that a single input is to be distributed to multiple database nodes for loading. The `SourceFactory` breaks the input into portions and assigns them to execution nodes. Each `Portion` consists of an offset into the input and a size. Vertica distributes the portions and their parameters to the execution nodes. A source factory running on each node produces a `UDSource` for the given portion.

The `UDParser` first determines where to start parsing:

- If the portion is the first one in the input, the parser advances to the offset and begins parsing.
- If the portion is not the first, the parser advances to the offset and then scans until it finds the end of a record. Because records can break across portions, parsing begins after the first record-end encountered.

The parser must complete a record, which might require it to read past the end of the portion. The parser is responsible for parsing all records that *begin* in the assigned portion, regardless of where they end. Most of this work occurs within the `process()` method of the parser.

Sometimes, a portion contains nothing to be parsed by its assigned node. For example, suppose you have a record that begins in portion 1, runs through all of portion 2, and ends in portion 3. The parser assigned to portion 1 parses the record, and the parser assigned to portion 3 starts after that record. The parser assigned to portion 2, however, has no record starting within its portion.

If the load also uses [Cooperative Parse](#), then after apportioning the load and before parsing, Vertica divides portions into chunks for parallel loading.

Implementing Apportioned Load

To implement apportioned load, perform the following actions in the source, the parser, and their factories.

In your `SourceFactory` subclass:

- Implement `isSourceApportionable()` and return true.
- Implement `plan()` to determine portion size, designate portions, and assign portions to execution nodes. To assign portions to particular executors, pass the information using the parameter `writer` on the plan context (`PlanContext::getWriter()`).
- Implement `prepareUDSources()`. Vertica calls this method on each execution node with the plan context created by the factory. This method returns the `UDSource` instances to be used for this node's assigned portions.
- If sources can take advantage of parallelism, you can implement `getDesiredThreads()` to request a number of threads for each source. See [SourceFactory Class](#) for more information about this method.

In your `UDSource` subclass, implement `process()` as you would for any other source, using the assigned portion. You can retrieve this portion with `getPortion()`.

In your `ParserFactory` subclass:

- Implement `isParserApportionable()` and return true.
- If your parser uses a `UDChunker` that supports apportioned load, implement `isChunkerApportionable()`.

In your `UDParser` subclass:

- Write your `UDParser` subclass to operate on portions rather than whole sources. You can do so by handling the stream states `PORTION_START` and `PORTION_END`, or by using the

ContinuousUDParser API. Your parser must scan for the beginning of the portion, find the first record boundary after that position, and parse to the end of the last record beginning in that portion. Be aware that this behavior might require that the parser read beyond the end of the portion.

- Handle the special case of a portion containing no record start by returning without writing any output.

In your UDChunker subclass, implement `alignPortion()`. See [Aligning Portions](#).

Example

The SDK provides a C++ example of apportioned load in the `ApportionLoadFunctions` directory:

- `FilePortionSource` is a subclass of `UDSource`.
- `DelimFilePortionParser` is a subclass of `ContinuousUDParser`.

Use these classes together. You could also use `FilePortionSource` with the built-in delimited parser.

The following example shows how you can load the libraries and create the functions in the database:

```
=> CREATE LIBRARY FilePortionSourceLib as '/home/dbadmin/FP.so';  
  
=> CREATE LIBRARY DelimFilePortionParserLib as '/home/dbadmin/Delim.so';  
  
=> CREATE SOURCE FilePortionSource AS  
LANGUAGE 'C++' NAME 'FilePortionSourceFactory' LIBRARY FilePortionSourceLib;  
  
=> CREATE PARSE DelimFilePortionParser AS  
LANGUAGE 'C++' NAME 'DelimFilePortionParserFactory' LIBRARY DelimFilePortionParserLib;
```

The following example shows how you can use the source and parser to load data:

```
=> COPY t WITH SOURCE FilePortionSource(file='g1/*.dat') PARSE DelimFilePortionParser(delimiter =  
'|',  
record_terminator = '~');
```

Combining Cooperative Parse and Apportioned Load

You can enable both [Cooperative Parse](#) and [Apportioned Load](#) in the same parser, allowing Vertica to decide how to load data.

Deciding How to Divide a Load

Vertica uses apportioned load, where possible, at query-planning time. It decides whether to also use cooperative parse at execution time.

Apportioned load requires `SourceFactory` support. Given a suitable `UDSource`, at planning time Vertica calls the `isParserApportionable()` method on the `ParserFactory`. If this method returns `true`, Vertica apportions the load.

If `isParserApportionable()` returns `false` but `isChunkerApportionable()` returns `true`, then a chunker is available for cooperative parse and that chunker supports apportioned load. Vertica apportions the load.

If neither of these methods returns `true`, then Vertica does not apportion the load.

At execution time, Vertica first checks whether the load is running in unfenced mode and proceeds only if it is. Cooperative parse is not supported in fenced mode.

If the load is not apportioned, and more than one thread is available, Vertica uses cooperative parse.

If the load is apportioned, and exactly one thread is available, Vertica uses cooperative parse if and only if the parser is not apportionable. In this case, the chunker is apportionable but the parser is not.

If the load is apportioned, and more than one thread is available, and the chunker is apportionable, Vertica uses cooperative parse.

If Vertica uses cooperative parse but `prepareChunker()` does not return a `UDChunker` instance, Vertica reports an error.

Executing Apportioned, Cooperative Loads

If a load uses both apportioned load and cooperative parse, Vertica uses the `SourceFactory` to break the input into portions. It then assigns the portions to execution nodes. See [How Vertica Apportions a Load](#).

On the execution node, Vertica calls the chunker's `alignPortion()` method to align the input with portion boundaries. (This step is skipped for the first portion, which by definition is already aligned at the beginning.) This step is necessary because a parser using apportioned load sometimes has to read beyond the end of the portion, so a chunker needs to find the end point.

After aligning the portion, Vertica calls the chunker's `process()` method repeatedly. See [How Vertica Divides a Load](#).

The chunks found by the chunker are then sent to the parser's `process()` method for processing in the usual way.

Continuous Load

The `ContinuousUDSource`, `ContinuousUDFilter`, and `ContinuousUDParser` classes allow you to write and process data as needed instead of having to iterate through the data.

Each class includes the following functions:

- `initialize()` - Invoked before `run()`. You can optionally override this function to perform setup and initialization.
- `run()` - Processes the data.
- `deinitialize()` - Invoked after `run()` has returned. You can optionally override this function to perform tear-down and destruction.

Do not override the `setup()`, `process()`, and `destroy()` functions that are inherited from parent classes.

You can use the `yield()` function to yield control back to the server during idle or busy loops so the server can check for status changes or query cancellations.

These three classes use associated `ContinuousReader` and `ContinuousWriter` classes to read input data and write output data.

C++ API

This section provides APIs and examples for the C++ API for UDLs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

Requirements for C++ UDLs

C++ UDLs:

- Can run in [Fenced Mode](#). Vertica enables fenced mode by default when you create a [source](#), [filter](#), or [parser](#) function, in Vertica unless you explicitly state otherwise.

- Must not permit an exception to be passed back to Vertica. Doing so could lead to issues such as memory leaks caused by the memory allocated by the exception never being freed. Your UDL should always contain a top-level try-catch block to catch any stray exceptions caused by your code or libraries that your code calls.
- Must properly free any resources that the UDL function allocates. Even a single byte of allocated memory that is not freed can become an issue in a UDL that is called over millions of rows. Instead of allocating memory directly, your function should use the memory allocation macros in the Vertica SDK. See [Allocating Resources for UDxs](#) for details.

The header files that define the majority of classes and methods are `VerticaUDx.h` and `VerticaUD1.h`. These header files, along with the main `Vertica.h` header file, are available in `/opt/vertica/sdk/include`.

Source Classes

See [User-Defined Source](#) for general information about implementing the `UDSource` and `SourceFactory` classes. This section describes information that is specific to the C++ API.

UDSource API

The API provides the following methods for extension by subclasses:

```
virtual void setup (ServerInterface &srvInterface);  
  
virtual StreamState process (ServerInterface &srvInterface,  
                             DataBuffer &output)=0;  
  
virtual void destroy (ServerInterface &srvInterface);  
  
virtual vint getSize ();  
  
virtual std::string getUri ();
```

ContinuousUDSource API

The `ContinuousUDSource` class extends `UDSource` and adds the following methods for extension by subclasses:

```
virtual void initialize (ServerInterface &srvInterface);  
  
virtual void run ();  
  
virtual void deinitialize (ServerInterface &srvInterface);
```

SourceFactory API

The API provides the following methods for extension by subclasses:

```
virtual void plan (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtx);

// must implement exactly one of prepareUDSources() or prepareUDSourcesExecutor()
virtual std::vector< UDSOURCE * > prepareUDSources (ServerInterface &srvInterface,
    NodeSpecifyingPlanContext &planCtx);

virtual std::vector< UDSOURCE * > prepareUDSourcesExecutor (ServerInterface &srvInterface,
    ExecutorPlanContext &planCtx);

virtual void getParameterType (ServerInterface &srvInterface,
    SizedColumnTypes &parameterTypes);

virtual bool isSourceApportionable ();

ssize_t getDesiredThreads(ServerInterface &srvInterface,
    ExecutorPlanContext &planContext);
```

After creating your SourceFactory, you must register it with the RegisterFactory macro.

Filter Classes

This section describes information that is specific to the C++ API. See [User-Defined Filter](#) for general information about implementing the UDFilter and FilterFactory classes.

UDFilter API

The API provides the following methods for extension by subclasses:

```
virtual void setup (ServerInterface & srvInterface);

virtual StreamState process (ServerInterface & srvInterface, DataBuffer &input,
    InputState input_state, DataBuffer &output)=0;

virtual void destroy (ServerInterface &srvInterface);
```

ContinuousUDFilter API

The ContinuousUDFilter class extends UDFilter and adds the following methods for extension by subclasses:

```
virtual void initialize (ServerInterface &srvInterface);

virtual void run ();
```

```
virtual void deinitialize (ServerInterface &srvInterface);
```

FilterFactory API

The API provides the following methods for extension by subclasses:

```
virtual void plan (ServerInterface &srvInterface, PlanContext &planCtxt);  
virtual UDFilter * prepare (ServerInterface &srvInterface, PlanContext &planCtxt)=0;  
virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes);
```

After creating your `FilterFactory`, you must register it with the `RegisterFactory` macro.

Parser Classes

This section describes information that is specific to the C++ API. See [User-Defined Parser](#) for general information about implementing the `UDParser` and `ParserFactory` classes.

UDParser API

The API provides the following methods for extension by subclasses:

```
virtual void setup (ServerInterface &srvInterface, SizedColumnTypes &returnType);  
virtual StreamState process (ServerInterface &srvInterface, DataBuffer &input, InputState input_state)=0;  
virtual void destroy (ServerInterface &srvInterface, SizedColumnTypes &returnType);  
virtual RejectedRecord getRejectedRecord ();
```

Rejecting Records

The `process()` method might need to reject input. To reject data, you must do two things:

- Create a `getRejectedRecord()` method on your `UDParser` subclass that returns an object of type `Vertica::RejectedRecord`. This record contains the data that you want to reject, a string describing the reason for rejection, the size of the data, and the terminator string. See `RejectedRecord` in `VerticaUDI.h` or the SDK documentation for

details.

- Reject a row by returning REJECT from `process()`. Vertica then calls `getRejectedRecord()` to process the rejected record before the next call to `process()`.

You can fulfill these requirements by including code in your parser class such as:

```
Vertica::RejectedRecord myRejRec;  
Vertica::RejectedRecord getRejectedRecord() {  
    return myRejRec;  
}
```

In your `process()` method, add code such as:

```
if (some rejection condition) {  
    RejectedRecord rr("Bad Record!", "foo data", 8, "\n");  
    myRejRec = rr;  
    return Vertica::REJECT;  
}
```

ContinuousUDParser API

The `ContinuousUDParser` class extends `UDParser` and adds the following methods for extension by subclasses:

```
virtual void initialize (ServerInterface &srvInterface);  
  
virtual void run ();  
  
virtual void deinitialize (ServerInterface &srvInterface);
```

UDChunker API

The `UDChunker` API provides the following methods for extension by subclasses:

```
virtual void setup (ServerInterface &srvInterface,  
                  SizedColumnTypes &returnType);  
  
virtual StreamState alignPortion (ServerInterface &srvInterface,  
                                 DataBuffer &input, InputState state);  
  
virtual StreamState process (ServerInterface &srvInterface,  
                             DataBuffer &input, InputState input_state)=0;  
  
virtual void destroy (ServerInterface &srvInterface,  
                    SizedColumnTypes &returnType);
```

ParserFactory API

The API provides the following methods for extension by subclasses:

```
virtual void plan (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt);

virtual UParser * prepare (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader,
    PlanContext &planCtxt, const SizedColumnTypes &returnType)=0;

virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes);

virtual void getParserReturnType (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader,
    PlanContext &planCtxt, const SizedColumnTypes &argTypes,
    SizedColumnTypes &returnType);

virtual bool isParserApportionable ();

// C++ API only:
virtual bool isChunkerApportionable (ServerInterface &srvInterface);

virtual UDChunker * prepareChunker (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader,
    PlanContext &planCtxt, const SizedColumnTypes &returnType);
```

If you are using [Apportioned Load](#) to divide a single input into multiple load streams, implement `isParserApportionable()` and/or `isChunkerApportionable()` and return true. Returning true from these methods does not guarantee that Vertica *will* apportion the load. However, returning false from both indicates that it will not try to do so.

If you are using [Cooperative Parse](#), implement `prepareChunker()` and return an instance of your UDChunker subclass. Cooperative parse is supported only for the C++ API.

Vertica calls the `prepareChunker()` method *only* for unfenced functions. This method is not available when you use the function in fenced mode.

If you want your chunker to be available for apportioned load, implement `isChunkerApportionable()` and return true.

After creating your ParserFactory, you must register it with the RegisterFactory macro.

Source Example: CurlSource

The CurlSource example allows you to use cURL to open and read in a file over HTTP. The example provided is part of:

```
/opt/vertica/sdk/examples/SourceFunctions/cURL.cpp.
```

Parser Implementation

This example uses the helper library available in `/opt/vertica/sdk/examples/HelperLibraries/`.

`CurlSource` loads the data in chunks. If the parser encounters an `EndOfFile` marker, then the `process()` method returns `DONE`. Otherwise, the method returns `OUTPUT_NEEDED` and processes another chunk of data. The functions included in the helper library (such as `url_fread()` and `url_fopen()`) are based on examples that come with the `libcurl` library. For an example, see <http://curl.haxx.se/libcurl/c/fopen.html>.

The `setup()` function opens a file handle and the `destroy()` function closes it. Both use functions from the helper library.

```
class CurlSource : public UDSOURCE {private:
    URL_FILE *handle;
    std::string url;
    virtual StreamState process(ServerInterface &srvInterface, DataBuffer &output) {
        output.offset = url_fread(output.buf, 1, output.size, handle);
        return url_feof(handle) ? DONE : OUTPUT_NEEDED;
    }
public:
    CurlSource(std::string url) : url(url) {}
    void setup(ServerInterface &srvInterface) {
        handle = url_fopen(url.c_str(), "r");
    }
    void destroy(ServerInterface &srvInterface) {
        url_fclose(handle);
    }
};
```

Factory Implementation

`CurlSourceFactory` produces `CurlSource` instances.

```
class CurlSourceFactory : public SourceFactory {public:
    virtual void plan(ServerInterface &srvInterface,
        NodeSpecifyingPlanContext &planCtx) {
        std::vector<std::string> args = srvInterface.getParamReader().getParamNames();
        /* Check parameters */
        if (args.size() != 1 || find(args.begin(), args.end(), "url") == args.end()) {
            vt_report_error(0, "You must provide a single URL.");
        }
        /* Populate planData */
        planCtx.getWriter().getStringRef("url").copy(
            srvInterface.getParamReader().getStringRef("url"));

        /* Assign Nodes */
        std::vector<std::string> executionNodes = planCtx.getClusterNodes();
        while (executionNodes.size() > 1) executionNodes.pop_back();
        // Only run on the first node in the list.
    }
};
```

```
    planCtxt.setTargetNodes(executionNodes);
  }
  virtual std::vector<UDSource*> prepareUDSources(ServerInterface &srvInterface,
    NodeSpecifyingPlanContext &planCtxt) {
    std::vector<UDSource*> retVal;
    retVal.push_back(vt_createFuncObj(srvInterface.allocator, CurlSource,
      planCtxt.getReader().getStringRef("url").str()));
    return retVal;
  }
  virtual void getParameterType(ServerInterface &srvInterface,
    SizedColumnTypes &parameterTypes) {
    parameterTypes.addVarchar(65000, "url");
  }
};
RegisterFactory(CurlSourceFactory);
```

Source Example: Concurrent Load

The `FilePortionSource` example demonstrates the use of concurrent load. This example is a refinement of the `FileSource` example. Each input file is divided into portions and distributed to `FilePortionSource` instances. The source accepts a list of offsets at which to break the input into portions; if offsets are not provided, the source divides the input dynamically.

Concurrent load is handled in the factory, so this discussion focuses on `FilePortionSourceFactory`. The full code for the example is located in `/opt/vertica/sdk/examples/ApportionLoadFunctions`. The distribution also includes a Java version of this example.

Loading and Using the Example

Load and use the `FilePortionSource` example as follows.

```
=> CREATE LIBRARY FilePortionLib AS '/home/dbadmin/FP.so';

=> CREATE SOURCE FilePortionSource AS LANGUAGE 'C++'
-> NAME 'FilePortionSourceFactory' LIBRARY FilePortionLib;

=> COPY t WITH SOURCE FilePortionSource(file='g1/*.dat', nodes='initiator,e0,e1', offsets =
'0,380000,820000');

=> COPY t WITH SOURCE FilePortionSource(file='g2/*.dat', nodes='e0,e1,e2', local_min_portion_size =
2097152);
```

Implementation

Concurrent load affects the source factory in two places, `getDesiredThreads()` and `prepareUDSourcesExecutor()`.

getDesiredThreads()

The `getDesiredThreads()` member function determines the number of threads to request. Vertica calls this member function on each executor node before calling `prepareUDSourcesExecutor()`.

The function begins by breaking an input file path, which might be a glob, into individual paths. This discussion omits those details. If apportioned load is not being used, then the function allocates one source per file.

```
virtual ssize_t getDesiredThreads(ServerInterface &srvInterface,
    ExecutorPlanContext &planCtx) {
    const std::string filename = srvInterface.getParamReader().getStringRef("file").str();

    std::vector<std::string> paths;
    // expand the glob - at least one thread per source.
    ...

    // figure out how to assign files to sources
    const std::string nodeName = srvInterface.getCurrentNodeName();
    const size_t nodeId = planCtx.getWriter().getIntRef(nodeName);
    const size_t numNodes = planCtx.getTargetNodes().size();

    if (!planCtx.canApportionSource()) {
        /* no apportioning, so the number of files is the final number of sources */
        std::vector<std::string> *expanded =
            vt_createFuncObject<std::vector<std::string> >(srvInterface.allocator, paths);
        /* save expanded paths so we don't have to compute expansion again */
        planCtx.getWriter().setPointer("expanded", expanded);
        return expanded->size();
    }

    // ...
}
```

If the source can be apportioned, then `getDesiredThreads()` uses the offsets that were passed as arguments to the factory to divide the file into portions. It then allocates portions to available nodes. This function does not actually assign sources directly; this work is done to determine how many threads to request.

```
else if (srvInterface.getParamReader().containsParameter("offsets")) {
    // if the offsets are specified, then we will have a fixed number of portions per file.
}
```

```
// Round-robin assign offsets to nodes.
// ...

/* Construct the portions that this node will actually handle.
 * This isn't changing (since the offset assignments are fixed),
 * so we'll build the Portion objects now and make them available
 * to prepareUDSourcesExecutor() by putting them in the ExecutorContext.
 *
 * We don't know the size of the last portion, since it depends on the file
 * size. Rather than figure it out here we will indicate it with -1 and
 * defer that to prepareUDSourcesExecutor().
 */
std::vector<Portion> *portions =
    vt_createFuncObject<std::vector<Portion>>(srvInterface.allocator);

for (std::vector<size_t>::const_iterator offset = offsets.begin();
     offset != offsets.end(); ++offset) {
    Portion p(*offset);
    p.is_first_portion = (offset == offsets.begin());
    p.size = (offset + 1 == offsets.end() ? -1 : (*(offset + 1) - *offset));

    if ((offset - offsets.begin()) % numNodes == nodeId) {
        portions->push_back(p);
        srvInterface.log("FilePortionSource: assigning portion %ld: [offset = %lld, size = %lld]",
            offset - offsets.begin(), p.offset, p.size);
    }
}
```

The function now has all the portions and thus the number of portions:

```
planCtx.getWriter().setPointer("portions", portions);

/* total number of threads we want is the number of portions per file, which is fixed */
return portions->size() * expanded->size();
} // end of "offsets" parameter
```

If offsets were not provided, the function divides the file into portions dynamically, one portion per thread. This discussion omits the details of this computation. There is no point in requesting more threads than are available, so the function calls `getMaxAllowedThreads()` on the `PlanContext` (an argument to the function) to set an upper bound:

```
if (portions->size() >= planCtx.getMaxAllowedThreads()) {
    return paths.size();
}
```

See the full example for the details of how this function divides the file into portions.

This function uses the `vt_createFuncObject` template to create objects. Vertica calls the destructors of returned objects created using this macro, but it does not call destructors for other objects like vectors. You must call these destructors yourself to avoid memory leaks. In this example, these calls are made in `prepareUDSourcesExecutor()`.

prepareUDSourcesExecutor()

The `prepareUDSourcesExecutor()` member function, like `getDesiredThreads()`, has separate blocks of code depending on whether offsets are provided. In both cases, the function breaks input into portions and creates `UDSource` instances for them.

If the function is called with offsets, `prepareUDSourcesExecutor()` calls `prepareCustomizedPortions()`. This function follows.

```
/* prepare portions as determined via the "offsets" parameter */
void prepareCustomizedPortions(ServerInterface &srvInterface,
                               ExecutorPlanContext &planCtx,
                               std::vector<UDSource *> &sources,
                               const std::vector<std::string> &expandedPaths,
                               std::vector<Portion> &portions) {
  for (std::vector<std::string>::const_iterator filename = expandedPaths.begin();
       filename != expandedPaths.end(); ++filename) {
    /*
     * the "portions" vector contains the portions which were generated in
     * "getDesiredThreads"
     */
    const size_t fileSize = getFileSize(*filename);
    for (std::vector<Portion>::const_iterator portion = portions.begin();
         portion != portions.end(); ++portion) {
      Portion fportion(*portion);
      if (fportion.size == -1) {
        /* as described above, this means from the offset to the end */
        fportion.size = fileSize - portion->offset;
        sources.push_back(vt_createFuncObject<FilePortionSource>(srvInterface.allocator,
                                                                *filename, fportion));
      } else if (fportion.size > 0) {
        sources.push_back(vt_createFuncObject<FilePortionSource>(srvInterface.allocator,
                                                                *filename, fportion));
      }
    }
  }
}
```

If `prepareUDSourcesExecutor()` is called without offsets, then it must decide how many portions to create.

The base case is to use one portion per source. However, if extra threads are available, the function divides the input into more portions so that a source can process them concurrently. Then `prepareUDSourcesExecutor()` calls `prepareGeneratedPortions()` to create the portions. This function begins by calling `getLoadConcurrency()` on the plan context to find out how many threads are available.

```
void prepareGeneratedPortions(ServerInterface &srvInterface,
                              ExecutorPlanContext &planCtx,
                              std::vector<UDSource *> &sources,
```

```
        std::map<std::string, Portion> initialPortions) {  
  
    if ((ssize_t) initialPortions.size() >= planCtxt.getLoadConcurrency()) {  
        /* all threads will be used, don't bother splitting into portions */  
  
    for (std::map<std::string, Portion>::const_iterator file = initialPortions.begin();  
         file != initialPortions.end(); ++file) {  
        sources.push_back(vt_createFuncObject<FilePortionSource>(srvInterface.allocator,  
            file->first, file->second));  
    } // for  
    return;  
} // if  
  
// Now we can split files to take advantage of potentially-unused threads.  
// First sort by size (descending), then we will split the largest first.  
  
// details elided...  
  
}
```

For More Information

See the source code for the full implementation of this example.

Filter Example: Converting Encoding

The following example shows how you can convert encoding for a file from one type to another by converting UTF-16 encoded data to UTF-8. You can find this example in the SDK at `/opt/vertica/sdk/examples/FilterFunctions/IConverter.cpp`.

Filter Implementation

```
class Iconverter : public UDFilter{  
private:  
    std::string fromEncoding, toEncoding;  
    iconv_t cd; // the conversion descriptor opened  
    uint converted; // how many characters have been converted  
protected:  
    virtual StreamState process(ServerInterface &srvInterface, DataBuffer &input,  
        InputState input_state, DataBuffer &output)  
    {  
        char *input_buf = (char *)input.buf + input.offset;  
        char *output_buf = (char *)output.buf + output.offset;  
        size_t inBytesLeft = input.size - input.offset, outBytesLeft = output.size - output.offset;  
        // end of input  
        if (input_state == END_OF_FILE && inBytesLeft == 0)  
        {  
            // Gnu libc iconv doc says, it is good practice to finalize the  
            // outbuffer for stateful encodings (by calling with null inbuffer).  
        }  
    }  
};
```

```
//
// http://www.gnu.org/software/libc/manual/html_node/Generic-Conversion-Interface.html
iconv(cd, NULL, NULL, &output_buf, &outBytesLeft);
// output buffer can be updated by this operation
output.offset = output.size - outBytesLeft;
return DONE;
}
size_t ret = iconv(cd, &input_buf, &inBytesLeft, &output_buf, &outBytesLeft);
// if conversion is successful, we ask for more input, as input has not reached EOF.
StreamState retStatus = INPUT_NEEDED;
if (ret == (size_t)(-1))
{
    // seen an error
    switch (errno)
    {
    case E2BIG:
        // input size too big, not a problem, ask for more output.
        retStatus = OUTPUT_NEEDED;
        break;
    case EINVAL:
        // input stops in the middle of a byte sequence, not a problem, ask for more input
        retStatus = input_state == END_OF_FILE ? DONE : INPUT_NEEDED;
        break;
    case EILSEQ:
        // invalid sequence seen, throw
        // TODO: reporting the wrong byte position
        vt_report_error(1, "Invalid byte sequence when doing %u-th conversion", converted);
    case EBADF:
        // something wrong with descriptor, throw
        vt_report_error(0, "Invalid descriptor");
    default:
        vt_report_error(0, "Uncommon Error");
        break;
    }
}
else converted += ret;
// move position pointer
input.offset = input.size - inBytesLeft;
output.offset = output.size - outBytesLeft;
return retStatus;
}
public:
Iconverter(const std::string &from, const std::string &to)
: fromEncoding(from), toEncoding(to), converted(0)
{
    // note "to encoding" is first argument to iconv...
    cd = iconv_open(to.c_str(), from.c_str());
    if (cd == (iconv_t)(-1))
    {
        // error when creating converters.
        vt_report_error(0, "Error initializing iconv: %m");
    }
}
~Iconverter()
{
    // free iconv resources;
    iconv_close(cd);
}
};
```

Factory Implementation

```
class IconverterFactory : public FilterFactory{
public:
    virtual void plan(ServerInterface &srvInterface,
        PlanContext &planCtxt) {
        std::vector<std::string> args = srvInterface.getParamReader().getParamNames();
        /* Check parameters */
        if (!(args.size() == 0 ||
            (args.size() == 1 && find(args.begin(), args.end(), "from_encoding")
                != args.end()) || (args.size() == 2
                && find(args.begin(), args.end(), "from_encoding") != args.end()
                && find(args.begin(), args.end(), "to_encoding") != args.end())) {
            vt_report_error(0, "Invalid arguments. Must specify either no arguments, or "
                "'from_encoding' alone, or 'from_encoding' and 'to_encoding'.");
        }
        /* Populate planData */
        // By default, we do UTF16->UTF8, and x->UTF8
        VString from_encoding = planCtxt.getWriter().getStringRef("from_encoding");
        VString to_encoding = planCtxt.getWriter().getStringRef("to_encoding");
        from_encoding.copy("UTF-16");
        to_encoding.copy("UTF-8");
        if (args.size() == 2)
        {
            from_encoding.copy(srvInterface.getParamReader().getStringRef("from_encoding"));
            to_encoding.copy(srvInterface.getParamReader().getStringRef("to_encoding"));
        }
        else if (args.size() == 1)
        {
            from_encoding.copy(srvInterface.getParamReader().getStringRef("from_encoding"));
        }
        if (!from_encoding.length()) {
            vt_report_error(0, "The empty string is not a valid from_encoding value");
        }
        if (!to_encoding.length()) {
            vt_report_error(0, "The empty string is not a valid to_encoding value");
        }
    }
    virtual UDFilter* prepare(ServerInterface &srvInterface,
        PlanContext &planCtxt) {
        return vt_createFuncObj(srvInterface.allocator, Iconverter,
            planCtxt.getReader().getStringRef("from_encoding").str(),
            planCtxt.getReader().getStringRef("to_encoding").str());
    }
    virtual void getParameterType(ServerInterface &srvInterface,
        SizedColumnTypes &parameterTypes) {
        parameterTypes.addVarchar(32, "from_encoding");
        parameterTypes.addVarchar(32, "to_encoding");
    }
};
RegisterFactory(IconverterFactory);
```

Parser Example: BasicIntegerParser

The `BasicIntegerParser` example parses a string of integers separated by non-numeric characters. For a version of this parser that uses continuous load, see [Parser Example: ContinuousIntegerParser](#).

Loading and Using the Example

Load and use the `BasicIntegerParser` example as follows.

```
=> CREATE LIBRARY BasicIntegerParserLib AS '/home/dbadmin/BIP.so';

=> CREATE PARSE BasicIntegerParser AS
LANGUAGE 'C++' NAME 'BasicIntegerParserFactory' LIBRARY BasicIntegerParserLib;

=> CREATE TABLE t (i integer);

=> COPY t FROM stdin WITH PARSE BasicIntegerParser();
0
1
2
3
4
5
\.
```

Implementation

The `BasicIntegerParser` class implements only the `process()` method from the API. (It also implements a helper method for type conversion.) This method processes each line of input, looking for numbers on each line. When it advances to a new line it moves the `input.offset` marker and checks the input state. It then writes the output.

```
virtual StreamState process(ServerInterface &srvInterface, DataBuffer &input,
                           InputState input_state) {
    // WARNING: This implementation is not trying for efficiency.
    // It is trying for simplicity, for demonstration purposes.

    size_t start = input.offset;
    const size_t end = input.size;

    do {
        bool found_newline = false;
        size_t numEnd = start;
        for (; numEnd < end; numEnd++) {
            if (input.buf[numEnd] < '0' || input.buf[numEnd] > '9') {
                found_newline = true;
                break;
            }
        }
    }
```

```
    }

    if (!found_newline) {
        input.offset = start;
        if (input_state == END_OF_FILE) {
            // If we're at end-of-file,
            // emit the last integer (if any) and return DONE.
            if (start != end) {
                writer->setInt(0, strtoint(input.buf + start, input.buf + numEnd));
                writer->next();
            }
            return DONE;
        } else {
            // Otherwise, we need more data.
            return INPUT_NEEDED;
        }
    }

    writer->setInt(0, strtoint(input.buf + start, input.buf + numEnd));
    writer->next();

    start = numEnd + 1;
} while (true);
};
```

In the factory, the `plan()` method is a no-op; there are no parameters to check. The `prepare()` method instantiates the parser using the macro provided by the SDK:

```
virtual UDParse* prepare(ServerInterface &srvInterface,
    PerColumnParamReader &perColumnParamReader,
    PlanContext &planCtxt,
    const SizedColumnTypes &returnType) {

    return vt_createFuncObject<BasicIntegerParser>(srvInterface.allocator);
}
```

The `getParserReturnType()` method declares the single output:

```
virtual void getParserReturnType(ServerInterface &srvInterface,
    PerColumnParamReader &perColumnParamReader,
    PlanContext &planCtxt,
    const SizedColumnTypes &argTypes,
    SizedColumnTypes &returnType) {
    // We only and always have a single integer column
    returnType.addInt(argTypes.getColumn(0));
}
```

As for all UDxs written in C++, the example ends by registering its factory:

```
RegisterFactory(BasicIntegerParserFactory);
```

Parser Example: ContinuousIntegerParser

The `ContinuousIntegerParser` example is a variation of `BasicIntegerParser`. Both examples parse integers from input strings. `ContinuousIntegerParser` uses [Continuous Load](#) to read data.

Loading and Using the Example

Load the `ContinuousIntegerParser` example as follows.

```
=> CREATE LIBRARY ContinuousIntegerParserLib AS '/home/dbadmin/CIP.so';

=> CREATE PARSER ContinuousIntegerParser AS
LANGUAGE 'C++' NAME 'ContinuousIntegerParserFactory'
LIBRARY ContinuousIntegerParserLib;
```

Use it in the same way that you use `BasicIntegerParser`. See [Parser Example: BasicIntegerParser](#).

Implementation

`ContinuousIntegerParser` is a subclass of `ContinuousUDParser`. Subclasses of `ContinuousUDParser` place the processing logic in the `run()` method.

```
virtual void run() {

    // This parser assumes a single-column input, and
    // a stream of ASCII integers split by non-numeric characters.
    size_t pos = 0;
    size_t reserved = cr.reserve(pos+1);
    while (!cr.isEof() || reserved == pos + 1) {
        while (reserved == pos + 1 && isdigit(*ptr(pos))) {
            pos++;
            reserved = cr.reserve(pos + 1);
        }

        std::string st(ptr(), pos);
        writer->setInt(0, strToInt(st));
        writer->next();

        while (reserved == pos + 1 && !isdigit(*ptr(pos))) {
            pos++;
            reserved = cr.reserve(pos + 1);
        }
        cr.seek(pos);
        pos = 0;
        reserved = cr.reserve(pos + 1);
    }
}
```

```
};
```

For a more complex example of a ContinuousUDParser, see [ExampleDelimitedParser](#) in the examples. (See [Downloading and Running UDx Example Code.](#))

[ExampleDelimitedParser](#) uses a chunker; see [Chunker Example: Delimited Parser and Chunker.](#)

Chunker Example: Delimited Parser and Chunker

The [ExampleDelimitedUDChunker](#) class divides an input at delimiter characters. You can use this chunker with any parser that understands delimited input.

[ExampleDelimitedParser](#) is a [ContinuousUDParser](#) subclass that uses this chunker.

Loading and Using the Example

Load and use the example as follows.

```
=> CREATE LIBRARY ExampleDelimitedParserLib AS '/home/dbadmin/EDP.so';

=> CREATE PARSER ExampleDelimitedParser AS
  LANGUAGE 'C++' NAME 'DelimitedParserFrameworkExampleFactory'
  LIBRARY ExampleDelimitedParserLib;

=> COPY t FROM stdin WITH PARSER ExampleDelimitedParser();
0
1
2
3
4
5
6
7
8
9
\.
```

Chunker Implementation

This chunker supports apportioned load. The `alignPortion()` method finds the beginning of the first complete record in the current portion and aligns the input buffer with it. The record terminator is passed as an argument and set in the constructor.

```
StreamState ExampleDelimitedUDChunker::alignPortion(
    ServerInterface &srvInterface,
    DataBuffer &input, InputState state)
{
```

```
/* find the first record terminator. Its record belongs to the previous portion */
void *buf = reinterpret_cast<void *>(input.buf + input.offset);
void *term = memchr(buf, recordTerminator, input.size - input.offset);

if (term) {
    /* record boundary found. Align to the start of the next record */
    const size_t chunkSize = reinterpret_cast<size_t>(term) - reinterpret_cast<size_t>(buf);
    input.offset += chunkSize
        + sizeof(char) /* length of record terminator */;

    /* input.offset points at the start of the first complete record in the portion */
    return DONE;
} else if (state == END_OF_FILE || state == END_OF_PORTION) {
    return REJECT;
} else {
    VIAssert(state == START_OF_PORTION || state == OK);
    return INPUT_NEEDED;
}
}
```

The `process()` method has to account for chunks that span portion boundaries. If the previous call was at the end of a portion, the method set a flag. The code begins by checking for and handling that condition. The logic is similar to that of `alignPortion()`, so the example calls it to do part of the division.

```
StreamState ExampleDelimitedUDChunker::process(
    ServerInterface &srvInterface,
    DataBuffer &input,
    InputState input_state)
{
    const size_t termLen = 1;
    const char *terminator = &recordTerminator;

    if (pastPortion) {
        /*
         * Previous state was END_OF_PORTION, and the last chunk we will produce
         * extends beyond the portion we started with, into the next portion.
         * To be consistent with alignPortion(), that means finding the first
         * record boundary, and setting the chunk to be at that boundary.
         * Fortunately, this logic is identical to aligning the portion (with
         * some slight accounting for END_OF_FILE)!
         */
        const StreamState findLastTerminator = alignPortion(srvInterface, input);

        switch (findLastTerminator) {
            case DONE:
                return DONE;
            case INPUT_NEEDED:
                if (input_state == END_OF_FILE) {
                    /* there is no more input where we might find a record terminator */
                    input.offset = input.size;
                    return DONE;
                }
                return INPUT_NEEDED;
            default:
                VIAssert("Invalid return state from alignPortion()");
        }
    }
}
```

```
    return findLastTerminator;  
}
```

Now the method looks for the delimiter. If the input began at the end of a portion, it sets the flag.

```
size_t ret = input.offset, term_index = 0;  
for (size_t index = input.offset; index < input.size; ++index) {  
    const char c = input.buff[index];  
    if (c == terminator[term_index]) {  
        ++term_index;  
        if (term_index == termLen) {  
            ret = index + 1;  
            term_index = 0;  
        }  
        continue;  
    } else if (term_index > 0) {  
        index -= term_index;  
    }  
  
    term_index = 0;  
}  
  
if (input_state == END_OF_PORTION) {  
    /*  
     * Regardless of whether or not a record was found, the next chunk will extend  
     * into the next portion.  
     */  
    pastPortion = true;  
}
```

Finally, `process()` moves the input offset and returns.

```
// if we were able to find some rows, move the offset to point at the start of the next (potential) row, or end of block  
if (ret > input.offset) {  
    input.offset = ret;  
    return CHUNK_ALIGNED;  
}  
  
if (input_state == END_OF_FILE) {  
    input.offset = input.size;  
    return DONE;  
}  
  
return INPUT_NEEDED;  
}
```

Factory Implementation

The file `ExampleDelimitedParser.cpp` defines a factory that uses this `UDChunker`. The chunker supports apportioned load, so the factory implements `isChunkerApportionable()`:

```
virtual bool isChunkerApportionable(ServerInterface &srvInterface) {
    ParamReader params = srvInterface.getParamReader();
    if (params.containsParameter("disable_chunker") && params.getBoolRef("d\
isable_chunker")) {
        return false;
    } else {
        return true;
    }
}
```

The `prepareChunker()` method creates the chunker:

```
virtual UDChunker* prepareChunker(ServerInterface &srvInterface,
    PerColumnParamReader &perColumnParamReader,
    r,
    PlanContext &planCtx,
    const SizedColumnTypes &returnType)
{
    ParamReader params = srvInterface.getParamReader();
    if (params.containsParameter("disable_chunker") && params.getBoolRef("d\
isable_chunker")) {
        return NULL;
    }

    std::string recordTerminator("\n");

    ParamReader args(srvInterface.getParamReader());
    if (args.containsParameter("record_terminator")) {
        recordTerminator = args.getStringRef("record_terminator").str();
    }

    return vt_createFuncObject<ExampleDelimitedUDChunker>(srvInterface.allo\
cator,
        recordTerminator[0]);
}
```

Java API

The Vertica Java SDK supports developing UDLs. The Java SDK enables you to create sources, filters, and parsers, but not chunkers.

For information on setting up a Java development environment and compiling and packaging libraries, see [Developing with the Java SDK](#).

Source Classes

This section describes information that is specific to the Java API. See [User-Defined Source](#) for general information about implementing the `UDSource` and `SourceFactory` classes.

UDSource API

The API provides the following methods for extension by subclasses:

```
public void setup (ServerInterface srvInterface) throws UdfException;

public abstract StreamState process (ServerInterface srvInterface, DataBuffer output) throws UdfException;

public void destroy (ServerInterface srvInterface) throws UdfException;

public Integer getSize ();

public String getUri ();
```

SourceFactory API

The API provides the following methods for extension by subclasses:

```
public void plan (ServerInterface srvInterface, NodeSpecifyingPlanContext planCtxt)
    throws UdfException;

// must implement one overload of prepareUDSources()
public ArrayList< UDSource > prepareUDSources (ServerInterface srvInterface,
    NodeSpecifyingPlanContext planCtxt)
    throws UdfException;

public ArrayList< UDSource > prepareUDSources (ServerInterface srvInterface,
    ExecutorPlanContext planCtxt)
    throws UdfException;

public void getParameterType (ServerInterface srvInterface, SizedColumnTypes parameterTypes);

public boolean isSourceApportionable();

public int getDesiredThreads(ServerInterface srvInterface,
    ExecutorPlanContext planCtxt)
    throws UdfException;
```

Filter Classes

This section describes information that is specific to the Java API. See [User-Defined Filter](#) for general information about implementing the `UDFilter` and `FilterFactory` classes.

UDFilter API

The API provides the following methods for extension by subclasses:

```
public void setup (ServerInterface srvInterface) throws UdfException;

public abstract StreamState process (ServerInterface srvInterface, DataBuffer input,
    InputState input_state, DataBuffer output)
    throws UdfException;

public void destroy (ServerInterface srvInterface) throws UdfException;
```

FilterFactory API

The API provides the following methods for extension by subclasses:

```
public void plan (ServerInterface srvInterface, PlanContext planCtxt)
    throws UdfException;

public abstract UDFilter prepare (ServerInterface srvInterface, PlanContext planCtxt)
    throws UdfException;

public void getParameterType (ServerInterface srvInterface, SizedColumnTypes parameterTypes);
```

Parser Classes

This section describes information that is specific to the Java API. See [User-Defined Parser](#) for general information about implementing the `UDParser` and `ParserFactory` classes.

UDParser API

The API provides the following methods for extension by subclasses:

```
public void setup(ServerInterface srvInterface, SizedColumnTypes returnType)
    throws UdfException;

public abstract StreamState process(ServerInterface srvInterface,
    DataBuffer input, InputState input_state)
    throws UdfException, DestroyInvocation;

public void destroy(ServerInterface srvInterface, SizedColumnTypes returnType)
    throws UdfException;

public RejectedRecord getRejectedRecord() throws UdfException;
```

A `UDParser` uses a `StreamWriter` to write its output. `StreamWriter` provides methods for all the basic types, such as `setBooleanValue()`, `setStringValue()`, and so on. In the Java API this class also provides the `setValue()` method, which automatically sets the data type.

The methods described so far write single column values. `StreamWriter` also provides a method to write a complete row from a map. The `setRowFromMap()` method takes a map of column names and values and writes all the values into their corresponding columns. This method does not define new columns but instead writes values only to existing columns. The `JsonParser` example uses this method to write arbitrary JSON input. (See [Parser Example: JSON Parser](#).)

Note: The `setRowFromMap()` method does not automatically advance the input to the next line; you must call `next()`. You can thus read a row and then override selected column values.

`setRowsFromMap()` also populates any `VMap` ('__raw__') column of Flex Tables (see [Using Flex Tables](#)) with the entire provided map. For most cases, `setRowsFromMap()` is the appropriate way to populate a Flex Table. However, you can also generate a `VMap` value into a specified column using `setVMap()`, similar to other `setValue()` methods.

The `setRowFromMap()` method automatically coerces the input values into the types defined for those columns using an associated `TypeCoercion`. In most cases, using the default implementation (`StandardTypeCoercion`) is appropriate.

`TypeCoercion` uses policies to govern its behavior. For example, the `FAIL_INVALID_INPUT_VALUE` policy means invalid input is treated as an error instead of using a null value. Errors are caught and handled as rejections (see "Rejecting Rows" in [User-Defined Parser](#)). Policies also govern whether input that is too long is truncated. Use the `setPolicy()` method on the parser's `TypeCoercion` to set policies. See the API documentation for supported values.

You might need to customize type coercion beyond setting these policies. To do so, subclass one of the provided implementations of `TypeCoercion` and override the `asType()` methods. Such customization could be necessary if your parser reads objects that come from a third-party library. A parser handling geo-coordinates, for example, might override `asLong` to translate inputs like "40.4397N" into numbers. See the Vertica API documentation for a list of implementations.

ContinuousUDParser API

The `ContinuousUDParser` class extends `UDParser` and adds the following methods for extension by subclasses:

```
void initialize (ServerInterface srvInterface, SizedColumnTypes returnType);  
  
abstract void run () throws UdfException;  
  
void deinitialize (ServerInterface srvInterface, SizedColumnTypes returnType);
```

See the API documentation for additional utility methods.

ParserFactory API

The API provides the following methods for extension by subclasses:

```
public void plan(ServerInterface srvInterface, PerColumnParamReader perColumnParamReader, PlanContext planCtxt)
    throws UdfException;

public abstract UDFParser prepare(ServerInterface srvInterface, PerColumnParamReader perColumnParamReader,
    PlanContext planCtxt, SizedColumnTypes returnType)
    throws UdfException;

public void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);

public void getParserReturnType(ServerInterface srvInterface, PerColumnParamReader perColumnParamReader,
    PlanContext planCtxt, SizedColumnTypes argTypes, SizedColumnTypes returnType)
    throws UdfException;
```

Source Example: FileSource

The example shown in this section is a simple UDL Source function named `FileSource`. This function loads data from files stored on the host's file system (similar to the standard [COPY](#) statement). To call `FileSource`, you must supply a parameter named `file` that contains the absolute path to one or more files on the host file system. You can specify multiple files as a comma-separated list.

The `FileSource` function also accepts an optional parameter, named `nodes`, that indicates which nodes should load the files. If you do not supply this parameter, the function defaults to loading data on the initiator node only. Because this example is simple, the nodes load only the files from their own file systems. Any files in the `file` parameter must exist on all of the hosts in the `nodes` parameter. The `FileSource` UDSOURCE attempts to load all of the files in the `file` parameter on all of the hosts in the `nodes` parameter.

Generating Files

You can use the following Python script to generate files and distribute them to hosts in your Vertica cluster. With these files, you can experiment with the example UDSOURCE function. Running the function requires passwordless-SSH logins to copy the files to the other hosts. Therefore, you must run the script using the database administrator account on one of your database hosts.

```
#!/usr/bin/python
# Save this file as UDLDDataGen.py
import string
```

```
import random
import sys
import os

# Read in the dictionary file to provide random words. Assumes the words
# file is located in /usr/share/dict/words
wordFile = open("/usr/share/dict/words")
wordDict = []
for line in wordFile:
    if len(line) > 6:
        wordDict.append(line.strip())

MAXSTR = 4 # Maximum number of words to concatenate
NUMROWS = 1000 # Number of rows of data to generate
#FILEPATH = '/tmp/UDLdata.txt' # Final filename to use for UDL source
TMPFILE = '/tmp/UDLtemp.txt' # Temporary filename.

# Generate a random string by concatenating several words together. Max
# number of words set by MAXSTR
def randomWords():
    words = [random.choice(wordDict) for n in xrange(random.randint(1, MAXSTR))]
    sentence = " ".join(words)
    return sentence

# Create a temporary data file that will be moved to a node. Number of
# rows for the file is set by NUMROWS. Adds the name of the node which will
# get the file, to show which node loaded the data.
def generateFile(node):
    outFile = open(TMPFILE, 'w')
    for line in xrange(NUMROWS):
        outFile.write('{0}|{1}|{2}\n'.format(line,randomWords(),node))
    outFile.close()

# Copy the temporary file to a node. Only works if passwordless SSH login
# is enabled, which it is for the database administrator account on
# Vertica hosts.
def copyFile(fileName,node):
    os.system('scp "%s" "%s:%s"' % (TMPFILE, node, fileName) )

# Loop through the comma-separated list of nodes given in the first
# parameter, creating and copying data files whose full comma-separated
# paths are passed in the second parameter
for node in [x.strip() for x in sys.argv[1].split(',')]:
    for fileName in [y.strip() for y in sys.argv[2].split(',')]:
        print "generating file", fileName, "for", node
        generateFile(node)
        print "Copying file to",node
        copyFile(fileName,node)
```

You call this script by giving it a comma-separated list of hosts to receive the files and a comma-separated list of absolute paths of files to generate. For example:

```
python UDLDataGen.py v_vmart_node0001,v_vmart_node0002,v_vmart_node0003
/tmp/UDLdata01.txt,/tmp/UDLdata02.txt,\
UDLdata03.txt
```

This script generates files that contain a thousand rows of columns delimited with the pipe character (|). These columns contain an index value, a set of random words, and the node for which the file was generated, as shown in the following output sample:

```
0|megabits embanks|v_vmart_node0001
1|unneatly|v_vmart_node0001
2|self-precipitation|v_vmart_node0001
3|antihistamine scalados Vatter|v_vmart_node0001
```

Loading and Using the Example

Load and use the FileSource UDSOURCE as follows:

```
=> --Load library and create the source function
=> CREATE LIBRARY JavaLib AS '/home/dbadmin/JavaUDLib.jar'
-> LANGUAGE 'JAVA';
CREATE LIBRARY
=> CREATE SOURCE File as LANGUAGE 'JAVA' NAME
-> 'com.mycompany.UDL.FileSourceFactory' LIBRARY JavaLib;
CREATE SOURCE FUNCTION
=> --Create a table to hold the data loaded from files
=> CREATE TABLE t (i integer, text VARCHAR, node VARCHAR);
CREATE TABLE
=> -- Copy a single file from the currently host using the FileSource
=> COPY t SOURCE File(file='/tmp/UDLdata01.txt');
Rows Loaded
-----
          1000
(1 row)

=> --See some of what got loaded.
=> SELECT * FROM t WHERE i < 5 ORDER BY i;
 i |          text          | node
-----+-----+-----
 0 | megabits embanks      | v_vmart_node0001
 1 | unneatly              | v_vmart_node0001
 2 | self-precipitation    | v_vmart_node0001
 3 | antihistamine scalados Vatter | v_vmart_node0001
 4 | fate-menaced toilworn | v_vmart_node0001
(5 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE
=> -- Now load a file from three hosts. All of these hosts must have a file
=> -- named /tmp/UDLdata01.txt, each with different data
=> COPY t SOURCE File(file='/tmp/UDLdata01.txt',
-> nodes='v_vmart_node0001,v_vmart_node0002,v_vmart_node0003');
Rows Loaded
-----
          3000
(1 row)

=> --Now see what has been loaded
=> SELECT * FROM t WHERE i < 5 ORDER BY i,node ;
```

```

i |          text          | node
-----+-----
0 | megabits embanks      | v_vmart_node0001
0 | nimble-eyed undupability frowsier | v_vmart_node0002
0 | Circean nonrepellence nonnasality | v_vmart_node0003
1 | unneatly              | v_vmart_node0001
1 | floatmaker trabacolos hit-in      | v_vmart_node0002
1 | revelrous treatableness Halleck   | v_vmart_node0003
2 | self-precipitation      | v_vmart_node0001
2 | whipcords archipelagic protodonatan copycutter | v_vmart_node0002
2 | Paganalian geochemistry short-shucks | v_vmart_node0003
3 | antihistamine scalados Vatter    | v_vmart_node0001
3 | swordweed touristical subcommanders desalinized | v_vmart_node0002
3 | batboys                 | v_vmart_node0003
4 | fate-menaced toilworn   | v_vmart_node0001
4 | twice-wanted cirrocumulous       | v_vmart_node0002
4 | doon-head-clock        | v_vmart_node0003
(15 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE
=> --Now copy from several files on several hosts
=> COPY t SOURCE File(file='/tmp/UDLdata01.txt,/tmp/UDLdata02.txt,/tmp/UDLdata03.txt'
-> ,nodes='v_vmart_node0001,v_vmart_node0002,v_vmart_node0003');
Rows Loaded
-----
          9000
(1 row)

=> SELECT * FROM t WHERE i = 0 ORDER BY node ;
i |          text          | node
-----+-----
0 | Awolowo Mirabilis D'Amboise      | v_vmart_node0001
0 | sortieing Divisionism selfhypnotization | v_vmart_node0001
0 | megabits embanks                  | v_vmart_node0001
0 | nimble-eyed undupability frowsier | v_vmart_node0002
0 | thiaminase hieroglypher derogated soilborne | v_vmart_node0002
0 | aurigraphy crocket stenocranial   | v_vmart_node0002
0 | Khulna pelmets                    | v_vmart_node0003
0 | Circean nonrepellence nonnasality | v_vmart_node0003
0 | matterate protarsal               | v_vmart_node0003
(9 rows)

```

Parser Implementation

The following code shows the source of the `FileSource` class that reads a file from the host file system. The constructor, which is called by `FileSourceFactory.prepareUDSources()`, gets the absolute path for the file containing the data to be read. The `setup()` method opens the file and the `destroy()` method closes it. The `process()` method reads from the file into a buffer provided by the instance of the `DataBuffer` class passed to it as a parameter. If the read operation filled the output buffer, it returns `OUTPUT_NEEDED`. This value tells Vertica to call the method again after the next stage of the load has processed the output buffer. If the read did not fill the output buffer, then `process()` returns `DONE` to indicate it has finished processing the data source.

```
package com.mycompany.UDL;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.RandomAccessFile;

import com.vertica.sdk.DataBuffer;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.State.StreamState;
import com.vertica.sdk.UDSource;
import com.vertica.sdk.UdfException;

public class FileSource extends UDSource {

    private String filename; // The file for this UDSource to read
    private RandomAccessFile reader; // handle to read from file

    // The constructor just stores the absolute filename of the file it will
    // read.
    public FileSource(String filename) {
        super();
        this.filename = filename;
    }

    // Called before Vertica starts requesting data from the data source.
    // In this case, setup needs to open the file and save to the reader
    // property.
    @Override
    public void setup(ServerInterface srvInterface ) throws UdfException{
        try {
            reader = new RandomAccessFile(new File(filename), "r");
        } catch (FileNotFoundException e) {
            // In case of any error, throw a UdfException. This will terminate
            // the data load.
            String msg = e.getMessage();
            throw new UdfException(0, msg);
        }
    }

    // Called after data has been loaded. In this case, close the file handle.
    @Override
    public void destroy(ServerInterface srvInterface ) throws UdfException {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                String msg = e.getMessage();
                throw new UdfException(0, msg);
            }
        }
    }

    @Override
    public StreamState process(ServerInterface srvInterface, DataBuffer output)
        throws UdfException {

        // Read up to the size of the buffer provided in the DataBuffer.buf
```

```
// property. Here we read directly from the file handle into the
// buffer.
long offset;
try {
    offset = reader.read(output.buf,output.offset,
        output.buf.length-output.offset);
} catch (IOException e) {
    // Throw an exception in case of any errors.
    String msg = e.getMessage();
    throw new UdfException(0, msg);
}

// Update the number of bytes processed so far by the data buffer.
output.offset +=offset;

// See end of data source has been reached, or less data was read
// than can fit in the buffer
if(offset == -1 || offset < output.buf.length) {
    // No more data to read.
    return StreamState.DONE;
}else{
    // Tell Vertica to call again when buffer has been emptied
    return StreamState.OUTPUT_NEEDED;
}
}
}
```

Factory Implementation

The following code is a modified version of the example Java UDsource function provided in the Java UDx support package. You can find the full example in `/opt/vertica/sdk/examples/JavaUDx/UDLFuctions/com/vertica/JavaLibs/FileSourceFactory.java`. Its override of the `plan()` method verifies that the user supplied the required file parameter. If the user also supplied the optional nodes parameter, this method verifies that the nodes exist in the Vertica cluster. If there is a problem with either parameter, the method throws an exception to return an error to the user. If there are no issues with the parameters, the `plan()` method stores their values in the plan context object.

```
package com.mycompany.UDL;

import java.util.ArrayList;
import java.util.Vector;
import com.vertica.sdk.NodeSpecifyingPlanContext;
import com.vertica.sdk.ParamReader;
import com.vertica.sdk.ParamWriter;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.SizedColumnTypes;
import com.vertica.sdk.SourceFactory;
import com.vertica.sdk.UDSource;
import com.vertica.sdk.UdfException;

public class FileSourceFactory extends SourceFactory {
```

```
// Called once on the initiator host to do initial setup. Checks
// parameters and chooses which nodes will do the work.
@Override
public void plan(ServerInterface srvInterface,
    NodeSpecifyingPlanContext planCtx) throws UdfException {

    String nodes; // stores the list of nodes that will load data

    // Get copy of the parameters the user supplied to the UDSource
    // function call.
    ParamReader args = srvInterface.getParamReader();

    // A list of nodes that will perform work. This gets saved as part
    // of the plan context.
    ArrayList<String> executionNodes = new ArrayList<String>();

    // First, ensure the user supplied the file parameter
    if (args.containsParameter("file")) {
        // Without a file parameter, we cannot continue. Throw an
        // exception that will be caught by the Java UDX framework.
        throw new UdfException(0, "You must supply a file parameter");
    }

    // If the user specified nodes to read the file, parse the
    // comma-separated list and save. Otherwise, assume just the
    // Initiator node has the file to read.
    if (args.containsParameter("nodes")) {
        nodes = args.getString("nodes");

        // Get list of nodes in cluster, to ensure that the node the
        // user specified actually exists. The list of nodes is available
        // from the planCtx (plan context) object,
        ArrayList<String> clusterNodes = planCtx.getClusterNodes();

        // Parse the string parameter "nodes" which
        // is a comma-separated list of node names.
        String[] nodeNames = nodes.split(",");

        for (int i = 0; i < nodeNames.length; i++){
            // See if the node the user gave us actually exists
            if(clusterNodes.contains(nodeNames[i]))
                // Node exists. Add it to list of nodes.
                executionNodes.add(nodeNames[i]);
            else{
                // User supplied node that doesn't exist. Throw an
                // exception so the user is notified.
                String msg = String.format("Specified node '%s' but no " +
                    "node by that name is available. Available nodes " +
                    "are \"%s\".",
                    nodeNames[i], clusterNodes.toString());
                throw new UdfException(0, msg);
            }
        }
    }
    // User did not supply a list of node names. Assume the initiator
    // is the only host that will read the file. The srvInterface
    // instance passed to this method has a getter for the current
    // node.
    executionNodes.add(srvInterface.getCurrentNodeName());
}
```

```
// Set the target node(s) in the plan context
planCtxt.setTargetNodes(executionNodes);

// Set parameters for each node reading data that tells it which
// files it will read. In this simple example, just tell it to
// read all of the files the user passed in the file parameter
String files = args.getString("file");

// Get object to write parameters into the plan context object.
ParamWriter nodeParams = planCtxt.getWriter();

// Loop through list of execution nodes, and add a parameter to plan
// context named for each node performing the work, which tells it the
// list of files it will process. Each node will look for a
// parameter named something like "filesForv_vmart_node0002" in its
// prepareUDSources() method.
for (int i = 0; i < executionNodes.size(); i++) {
    nodeParams.setString("filesFor" + executionNodes.get(i), files);
}
}

// Called on each host that is reading data from a source. This method
// returns an array of UDSOURCE objects that process each source.
@Override
public ArrayList<UDSource> prepareUDSources(ServerInterface srvInterface,
    NodeSpecifyingPlanContext planCtxt) throws UdfException {

    // An array to hold the UDSOURCE subclasses that we instantiate
    ArrayList<UDSource> retVal = new ArrayList<UDSource>();

    // Get the list of files this node is supposed to process. This was
    // saved by the plan() method in the plancontext
    String myName = srvInterface.getCurrentNodeName();
    ParamReader params = planCtxt.getReader();
    String fileNames = params.getString("filesFor" + myName);

    // Note that you can also be lazy and directly grab the parameters
    // the user passed to the UDSOURCE function in the COPY statement directly
    // by getting parameters from the ServerInterface object. I.e.:

    //String fileNames = srvInterface.getParamReader().getString("file");

    // Split comma-separated list into a single list.
    String[] fileList = fileNames.split(",");
    for (int i = 0; i < fileList.length; i++){
        // Instantiate a FileSource object (which is a subclass of UDSOURCE)
        // to read each file. The constructor for FileSource takes the
        // file name of the
        retVal.add(new FileSource(fileList[i]));
    }

    // Return the collection of FileSource objects. They will be called,
    // in turn, to read each of the files.
    return retVal;
}

// Declares which parameters that this factory accepts.
@Override
public void getParameterType(ServerInterface srvInterface,
```

```
        SizedColumnTypes parameterTypes) {
    parameterTypes.addVarchar(65000, "file");
    parameterTypes.addVarchar(65000, "nodes");
}
}
```

Filter Example: ReplaceCharFilter

The example in this section demonstrates creating a `UDFilter` that replaces any occurrences of a character in the input stream with another character in the output stream. This example is highly simplified and assumes the input stream is ASCII data.

Always remember that the input and output streams in a `UDFilter` are actually binary data. If you are performing character transformations using a `UDFilter`, convert the data stream from a string of bytes into a properly encoded string. For example, your input stream might consist of UTF-8 encoded text. If so, be sure to transform the raw binary being read from the buffer into a UTF string before manipulating it.

Loading and Using the Example

The example `UDFilter` has two required parameters. The `from_char` parameter specifies the character to be replaced, and the `to_char` parameter specifies the replacement character. Load and use the `ReplaceCharFilter` `UDFilter` as follows:

```
=> CREATE LIBRARY JavaLib AS '/home/dbadmin/JavaUDLib.jar'
->LANGUAGE 'JAVA';
CREATE LIBRARY
=> CREATE FILTER ReplaceCharFilter as LANGUAGE 'JAVA'
->name 'com.mycompany.UDL.ReplaceCharFilterFactory' library JavaLib;
CREATE FILTER FUNCTION
=> CREATE TABLE t (text VARCHAR);
CREATE TABLE
=> COPY t FROM STDIN WITH FILTER ReplaceCharFilter(from_char='a', to_char='z');
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Mary had a little lamb
>> a man, a plan, a canal, Panama
>> \.

=> SELECT * FROM t;
          text
-----
Mzry hzd z little lzmb
z mzn, z plzn, z cznzl, Pznmz
(2 rows)

=> --Calling the filter with incorrect parameters returns errors
=> COPY t from stdin with filter ReplaceCharFilter();
ERROR 3399: Failure in UDX RPC call InvokePlanUDL(): Error in User Defined Object [
```

```
ReplaceCharFilter], error code: 0  
com.vertica.sdk.UdfException: You must supply two parameters to ReplaceChar: 'from_char' and 'to_  
char'  
    at com.vertica.JavaLibs.ReplaceCharFilterFactory.plan(ReplaceCharFilterFactory.java:22)  
    at com.vertica.udxfence.UdxExecContext.planUDFilter(UDxExecContext.java:889)  
    at com.vertica.udxfence.UdxExecContext.planCurrentUDLType(UDxExecContext.java:865)  
    at com.vertica.udxfence.UdxExecContext.planUDL(UDxExecContext.java:821)  
    at com.vertica.udxfence.UdxExecContext.run(UDxExecContext.java:242)  
    at java.lang.Thread.run(Thread.java:662)
```

Parser Implementation

The `ReplaceCharFilter` class reads the data stream, replacing each occurrence of a user-specified character with another character.

```
package com.vertica.JavaLibs;  
  
import com.vertica.sdk.DataBuffer;  
import com.vertica.sdk.ServerInterface;  
import com.vertica.sdk.State.InputState;  
import com.vertica.sdk.State.StreamState;  
import com.vertica.sdk.UDFilter;  
  
public class ReplaceCharFilter extends UDFilter {  
  
    private byte[] fromChar;  
    private byte[] toChar;  
  
    public ReplaceCharFilter(String fromChar, String toChar){  
        // Stores the from char and to char as byte arrays. This is  
        // not a robust method of doing this, but works for this simple  
        // example.  
        this.fromChar= fromChar.getBytes();  
        this.toChar=toChar.getBytes();  
    }  
    @Override  
    public StreamState process(ServerInterface srvInterface, DataBuffer input,  
        InputState input_state, DataBuffer output) {  
  
        // Check if there is no more input and the input buffer has been completely  
        // processed. If so, filtering is done.  
        if (input_state == InputState.END_OF_FILE && input.buf.length == 0) {  
            return StreamState.DONE;  
        }  
  
        // Get current position in the input buffer  
        int offset = output.offset;  
  
        // Determine how many bytes to process. This is either until input  
        // buffer is exhausted or output buffer is filled  
        int limit = Math.min((input.buf.length - input.offset),  
            (output.buf.length - output.offset));  
  
        for (int i = input.offset; i < limit; i++) {  
            // This example just replaces each instance of from_char  
            // with to_char. It does not consider things such as multi-byte
```

```
// UTF-8 characters.
if (input.buf[i] == fromChar[0]) {
    output.buf[i+offset] = toChar[0];
} else {
    // Did not find from_char, so copy input to the output
    output.buf[i+offset]=input.buf[i];
}
}

input.offset += limit;
output.offset += input.offset;

if (input.buf.length - input.offset < output.buf.length - output.offset) {
    return StreamState.INPUT_NEEDED;
} else {
    return StreamState.OUTPUT_NEEDED;
}
}
}
```

Factory Implementation

`ReplaceCharFilterFactory` requires two parameters (`from_char` and `to_char`). The `plan()` method verifies that these parameters exist and are single-character strings. The method then stores them in the plan context. The `prepare()` method gets the parameter values and passes them to the `ReplaceCharFilter` objects, which it instantiates, to perform the filtering.

```
package com.vertica.JavaLibs;

import java.util.ArrayList;
import java.util.Vector;

import com.vertica.sdk.FilterFactory;
import com.vertica.sdk.PlanContext;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.SizedColumnTypes;
import com.vertica.sdk.UDFilter;
import com.vertica.sdk.UdfException;

public class ReplaceCharFilterFactory extends FilterFactory {

    // Run on the initiator node to perform varification and basic setup.
    @Override
    public void plan(ServerInterface srvInterface, PlanContext planCtx)
        throws UdfException {
        ArrayList<String> args =
            srvInterface.getParamReader().getParamNames();

        // Ensure user supplied two arguments
        if (!(args.contains("from_char") && args.contains("to_char"))) {
            throw new UdfException(0, "You must supply two parameters" +
                " to ReplaceChar: 'from_char' and 'to_char'");
        }
    }
}
```

```
// Verify that the from_char is a single character.
String fromChar = srvInterface.getParamReader().getString("from_char");
if (fromChar.length() != 1) {
    String message = String.format("Replacechar expects a single " +
        "character in the 'from_char' parameter. Got length %d",
        fromChar.length());
    throw new UdfException(0, message);
}

// Save the from character in the plan context, to be read by
// prepare() method.
planCtx.getWriter().setString("fromChar",fromChar);

// Ensure to character parameter is a single character
String toChar = srvInterface.getParamReader().getString("to_char");
if (toChar.length() != 1) {
    String message = String.format("Replacechar expects a single "
        + "character in the 'to_char' parameter. Got length %d",
        toChar.length());
    throw new UdfException(0, message);
}
// Save the to character in the plan data
planCtx.getWriter().setString("toChar",toChar);
}

// Called on every host that will filter data. Must instantiate the
// UDFilter subclass.
@Override
public UDFilter prepare(ServerInterface srvInterface, PlanContext planCtx)
    throws UdfException {
    // Get data stored in the context by the plan() method.
    String fromChar = planCtx.getWriter().getString("fromChar");
    String toChar = planCtx.getWriter().getString("toChar");

    // Instantiate a filter object to perform filtering.
    return new ReplaceCharFilter(fromChar, toChar);
}

// Describe the parameters accepted by this filter.
@Override
public void getParameterType(ServerInterface srvInterface,
    SizedColumnTypes parameterTypes) {
    parameterTypes.addVarchar(1, "from_char");
    parameterTypes.addVarchar(1, "to_char");
}
}
```

Parser Example: Numeric Text

This `NumericTextParser` example parses integer values spelled out in words rather than digits (for example "one two three" for one-hundred twenty three). The parser:

- Accepts a single parameter to set the character that separates columns in a row of data. The separator defaults to the pipe (|) character.
- Ignores extra spaces and the capitalization of the words used to spell out the digits.
- Recognizes the digits using the following words: zero, one, two, three, four, five, six, seven, eight, nine.
- Assumes that the words spelling out an integer are separated by at least one space.
- Rejects any row of data that cannot be completely parsed into integers.
- Generates an error, if the output table has a non-integer column.

Loading and Using the Example

Load and use the parser as follows:

```
=> CREATE LIBRARY JavaLib AS '/home/dbadmin/JavaLib.jar' LANGUAGE 'JAVA';
CREATE LIBRARY

=> CREATE PARSER NumericTextParser AS LANGUAGE 'java'
->   NAME 'com.myCompany.UDParser.NumericTextParserFactory'
->   LIBRARY JavaLib;
CREATE PARSER FUNCTION
=> CREATE TABLE t (i INTEGER);
CREATE TABLE
=> COPY t FROM STDIN WITH PARSER NumericTextParser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> One
>> Two
>> One Two Three
>> \.
=> SELECT * FROM t ORDER BY i;
   i
-----
   1
   2
  123
(3 rows)

=> DROP TABLE t;
DROP TABLE
=> -- Parse multi-column input
=> CREATE TABLE t (i INTEGER, j INTEGER);
CREATE TABLE
=> COPY t FROM stdin WITH PARSER NumericTextParser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> One | Two
>> Two | Three
>> One Two Three | four Five Six
```

```
>> \.
=> SELECT * FROM t ORDER BY i;
  i | j
-----+-----
  1 |  2
  2 |  3
123 | 456
(3 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE
=> -- Use alternate separator character
=> COPY t FROM STDIN WITH PARSER NumericTextParser(separator='*');
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Five * Six
>> seven * eight
>> nine * one zero
>> \.
=> SELECT * FROM t ORDER BY i;
  i | j
-----+-----
  5 |  6
  7 |  8
  9 | 10
(3 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE

=> -- Rows containing data that does not parse into digits is rejected.
=> DROP TABLE t;
DROP TABLE
=> CREATE TABLE t (i INTEGER);
CREATE TABLE
=> COPY t FROM STDIN WITH PARSER NumericTextParser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> One Zero Zero
>> Two Zero Zero
>> Three Zed Zed
>> Four Zero Zero
>> Five Zed Zed
>> \.
SELECT * FROM t ORDER BY i;
  i
-----
 100
 200
 400
(3 rows)

=> -- Generate an error by trying to copy into a table with a non-integer column
=> DROP TABLE t;
DROP TABLE
=> CREATE TABLE t (i INTEGER, j VARCHAR);
CREATE TABLE
=> COPY t FROM STDIN WITH PARSER NumericTextParser();
vsq1:UDParse.sql:94: ERROR 3399: Failure in UDX RPC call
InvokeGetReturnTypeParser(): Error in User Defined Object [NumericTextParser],
```

```
error code: 0
com.vertica.sdk.UdfException: Column 2 of output table is not an Int
    at com.myCompany.UDParser.NumericTextParserFactory.getParserReturnType
      (NumericTextParserFactory.java:70)
    at com.vertica.udxfence.UDxExecContext.getReturnTypeParser(
      UDxExecContext.java:1464)
    at com.vertica.udxfence.UDxExecContext.getReturnTypeParser(
      UDxExecContext.java:768)
    at com.vertica.udxfence.UDxExecContext.run(UDxExecContext.java:236)
    at java.lang.Thread.run(Thread.java:662)
```

Parser Implementation

The following code implements the parser.

```
package com.myCompany.UDParser;

import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;

import com.vertica.sdk.DataBuffer;
import com.vertica.sdk.DestroyInvocation;
import com.vertica.sdk.RejectedRecord;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.State.InputState;
import com.vertica.sdk.State.StreamState;
import com.vertica.sdk.StreamWriter;
import com.vertica.sdk.UDParser;
import com.vertica.sdk.UdfException;

public class NumericTextParser extends UDParser {

    private String separator; // Holds column separator character

    // List of strings that we accept as digits.
    private List<String> numbers = Arrays.asList("zero", "one",
        "two", "three", "four", "five", "six", "seven",
        "eight", "nine");

    // Hold information about the last rejected row.
    private String rejectedReason;
    private String rejectedRow;

    // Constructor gets the separator character from the Factory's prepare()
    // method.
    public NumericTextParser(String sepparam) {
        super();
        this.separator = sepparam;
    }

    // Called to perform the actual work of parsing. Gets a buffer of bytes
    // to turn into tuples.
    @Override
    public StreamState process(ServerInterface srvInterface, DataBuffer input,
        InputState input_state) throws UdfException, DestroyInvocation {
```

```
int i=input.offset; // Current position in the input buffer
// Flag to indicate whether we just found the end of a row.
boolean lastCharNewline = false;
// Buffer to hold the row of data being read.
StringBuffer line = new StringBuffer();

//Continue reading until end of buffer.
for(; i < input.buf.length; i++){
    // Loop through input until we find a linebreak: marks end of row
    char inchar = (char) input.buf[i];
    // Note that this isn't a robust way to find rows. It should
    // accept a user-defined row separator. Also, the following
    // assumes ASCII line break methods, which isn't a good idea
    // in the UTF world. But it is good enough for this simple example.
    if (inchar != '\n' && inchar != '\r') {
        // Keep adding to a line buffer until a full row of data is read
        line.append(inchar);
        lastCharNewline = false; // Last character not a new line
    } else {
        // Found a line break. Process the row.
        lastCharNewline = true; // indicate we got a complete row
        // Update the position in the input buffer. This is updated
        // whether the row is successfully processed or not.
        input.offset = i+1;
        // Call procesRow to extract values and write tuples to the
        // output. Returns false if there was an error.
        if (!processRow(line)) {
            // Processing row failed. Save bad row to rejectedRow field
            // and return to caller indicating a rejected row.
            rejectedRow = line.toString();
            // Update position where we processed the data.
            return StreamState.REJECT;
        }
        line.delete(0, line.length()); // clear row buffer
    }
}

// At this point, process() has finished processing the input buffer.
// There are two possibilities: need to get more data
// from the input stream to finish processing, or there is
// no more data to process. If at the end of the input stream and
// the row was not terminated by a linefeed, it may need
// to process the last row.

if (input_state == InputState.END_OF_FILE && lastCharNewline) {
    // End of input and it ended on a newline. Nothing more to do
    return StreamState.DONE;
} else if (input_state == InputState.END_OF_FILE && !lastCharNewline) {
    // At end of input stream but didn't get a final newline. Need to
    // process the final row that was read in, then exit for good.
    if (line.length() == 0) {
        // Nothing to process. Done parsing.
        return StreamState.DONE;
    }
    // Need to parse the last row, not terminated by a linefeed. This
    // can occur if the file being read didn't have a final line break.
    if (processRow(line)) {
        return StreamState.DONE;
    } else {
```

```
        // Processing last row failed. Save bad row to rejectedRow field
        // and return to caller indicating a rejected row.
        rejectedRow = line.toString();
        // Tell Vertica the entire buffer was processed so it won't
        // call again to have the line processed.
        input.offset = input.buf.length;
        return StreamState.REJECT;
    }
} else {
    // Stream is not fully read, so tell Vertica to send more. If
    // process() didn't get a complete row before it hit the end of the
    // input buffer, it will end up re-processing that segment again
    // when more data is added to the buffer.
    return StreamState.INPUT_NEEDED;
}
}

// Breaks a row into columns, then parses the content of the
// columns. Returns false if there was an error parsing the
// row, in which case it sets the rejected row to the input
// line. Returns true if the row was successfully read.
private boolean processRow(StringBuffer line)
    throws UdfException, DestroyInvocation {
    String[] columns = line.toString().split(Pattern.quote(separator));
    // Loop through the columns, decoding their contents
    for (int col = 0; col < columns.length; col++) {
        // Call decodeColumn to extract value from this column
        Integer colval = decodeColumn(columns[col]);
        if (colval == null) {
            // Could not parse one of the columns. Indicate row should
            // be rejected.
            return false;
        }
        // Column parsed OK. Write it to the output. writer is a field
        // provided by the parent class. Since this parser only accepts
        // integers, there is no need to verify that data type of the parsed
        // data matches the data type of the column being written. In your
        // UDParsers, you may want to perform this verification.
        writer.setLong(col,colval);
    }
    // Done with the row of data. Advance output to next row.

    // Note that this example does not verify that all of the output columns
    // have values assigned to them. If there are missing values at the
    // end of a row, they get automatically get assigned a default value
    // (0 for integers). This isn't a robust solution. Your UDParser
    // should perform checks here to handle this situation and set values
    // (such as null) when appropriate.
    writer.next();
    return true; // Successfully processed the row.
}

// Gets a string with text numerals, i.e. "One Two Five Seven" and turns
// it into an integer value, i.e. 1257. Returns null if the string could not
// be parsed completely into numbers.
private Integer decodeColumn(String text) {
    int value = 0; // Hold the value being parsed.

    // Split string into individual words. Eat extra spaces.
    String[] words = text.toLowerCase().trim().split("\\s+");
```

```
// Loop through the words, matching them against the list of
// digit strings.
for (int i = 0; i < words.length; i++) {
    if (numbers.contains(words[i])) {
        // Matched a digit. Add the it to the value.
        int digit = numbers.indexOf(words[i]);
        value = (value * 10) + digit;
    } else {
        // The string didn't match one of the accepted string values
        // for digits. Need to reject the row. Set the rejected
        // reason string here so it can be incorporated into the
        // rejected reason object.
        //
        // Note that this example does not handle null column values.
        // In most cases, you want to differentiate between an
        // unparseable column value and a missing piece of input
        // data. This example just rejects the row if there is a missing
        // column value.
        rejectedReason = String.format(
            "Could not parse '%s' into a digit", words[i]);
        return null;
    }
}
return value;
}

// Vertica calls this method if the parser rejected a row of data
// to find out what went wrong and add to the proper logs. Just gathers
// information stored in fields and returns it in an object.
@Override
public RejectedRecord getRejectedRecord() throws UdfException {
    return new RejectedRecord(rejectedReason, rejectedRow.toCharArray(),
        rejectedRow.length(), "\n");
}
}
```

ParserFactory Implementation

The following code implements the parser factory.

`NumericTextParser` accepts a single optional parameter named `separator`. This parameter is defined in the `getParameterType()` method, and the `plan()` method stores its value. `NumericTextParser` outputs only integer values. Therefore, if the output table contains a column whose data type is not integer, the `getParserReturnType()` method throws an exception.

```
package com.myCompany.UDParser;

import java.util.regex.Pattern;

import com.vertica.sdk.ParamReader;
import com.vertica.sdk.ParamWriter;
import com.vertica.sdk.ParserFactory;
```

```
import com.vertica.sdk.PerColumnParamReader;
import com.vertica.sdk.PlanContext;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.SizedColumnTypes;
import com.vertica.sdk.UDPParser;
import com.vertica.sdk.UdfException;
import com.vertica.sdk.VerticaType;

public class NumericTextParserFactory extends ParserFactory {

    // Called once on the initiator host to check the parameters and set up the
    // context data that hosts performing processing will need later.
    @Override
    public void plan(ServerInterface srvInterface,
        PerColumnParamReader perColumnParamReader,
        PlanContext planCtx) {

        String separator = "|"; // assume separator is pipe character

        // See if a parameter was given for column separator
        ParamReader args = srvInterface.getParamReader();
        if (args.containsParameter("separator")) {
            separator = args.getString("separator");
            if (separator.length() > 1) {
                throw new UdfException(0,
                    "Separator parameter must be a single character");
            }
            if (Pattern.quote(separator).matches("[a-zA-Z]")) {
                throw new UdfException(0,
                    "Separator parameter cannot be a letter");
            }
        }

        // Save separator character in the Plan Data
        ParamWriter context = planCtx.getWriter();
        context.setString("separator", separator);
    }

    // Define the data types of the output table that the parser will return.
    // Mainly, this just ensures that all of the columns in the table which
    // is the target of the data load are integer.
    @Override
    public void getParserReturnType(ServerInterface srvInterface,
        PerColumnParamReader perColumnParamReader,
        PlanContext planCtx,
        SizedColumnTypes argTypes,
        SizedColumnTypes returnType) {

        // Get access to the output table's columns
        for (int i = 0; i < argTypes.getColumnCount(); i++) {
            if (argTypes.getColumnType(i).isInt()) {
                // Column is integer... add it to the output
                returnType.addInt(argTypes.getColumnName(i));
            } else {
                // Column isn't an int, so throw an exception.
                // Technically, not necessary since the
                // UDX framework will automatically error out when it sees a
                // discrepancy between the type in the target table and the
                // types declared by this method. Throwing this exception will
                // provide a clearer error message to the user.
            }
        }
    }
}
```

```
        String message = String.format(
            "Column %d of output table is not an Int", i + 1);
        throw new UdfException(0, message);
    }
}

// Instantiate the UDParser subclass named NumericTextParser. Passes the
// separator characetr as a paramter to the constructor.
@Override
public UDParser prepare(ServerInterface srvInterface,
    PerColumnParamReader perColumnParamReader, PlanContext planCtxt,
    SizedColumnTypes returnType) throws UdfException {
    // Get the separator character from the context
    String separator = planCtxt.getReader().getString("separator");
    return new NumericTextParser(separator);
}

// Describe the parameters accepted by this parser.
@Override
public void getParameterType(ServerInterface srvInterface,
    SizedColumnTypes parameterTypes) {
    parameterTypes.addVarchar(1, "separator");
}
}
```

Parser Example: JSON Parser

The JSON Parser consumes a stream of JSON objects. Each object must be well formed and on a single line in the input. Use line breaks to delimit the objects. The parser uses the field names as keys in a map, which become column names in the table. You can find the code for this example in `/opt/vertica/packages/flextable/examples`. This directory also contains an example data file.

This example uses the `setRowFromMap()` method to write data.

Loading and Using the Example

1. Load the library and define the JSON parser, using the third-party library (gson-2.2.4.jar). See the comments in `JsonParser.java` for a download URL.

```
=> CREATE LIBRARY json
-> AS '/opt/vertica/packages/flextable/examples/java/output/json.jar'
-> DEPENDS '/opt/vertica/bin/gson-2.2.4.jar' language 'java';
CREATE LIBRARY

=> CREATE PARSEr JsonParser AS LANGUAGE 'java'
-> NAME 'com.vertica.flex.JsonParserFactory' LIBRARY json;
CREATE PARSEr FUNCTION
```

2. Define a table, and then use the JSON parser to load data into that table.

```
=> CREATE TABLE mountains(name varchar(64), type varchar(32), height integer);
CREATE TABLE

=> COPY mountains FROM '/opt/vertica/packages/flextable/examples/mountains.json'
-> WITH PARSER JsonParser();
-[ RECORD 1 ]--
Rows Loaded | 2

=> SELECT * from mountains;
-[ RECORD 1 ]-----
name   | Everest
type   | mountain
height | 29029
-[ RECORD 2 ]-----
name   | Mt St Helens
type   | volcano
height |
```

The data file contains a value (`hike_safety`) that was not loaded because the table definition did not include that column. The data file follows:

```
{ "name": "Everest", "type":"mountain", "height": 29029, "hike_safety": 34.1 }
{ "name": "Mt St Helens", "type": "volcano", "hike_safety": 15.4 }
```

Implementation

The following code shows the `process()` method from `JsonParser.java`. The parser attempts to read the input into a `Map`. If the read is successful, the JSON Parser calls `setRowFromMap()`:

```
@Override
public StreamState process(ServerInterface srvInterface, DataBuffer input,
    InputState inputState) throws UdfException, DestroyInvocation {
    clearReject();
    StreamWriter output = getStreamWriter();

    while (input.offset < input.buf.length) {
        ByteBuffer lineBytes = consumeNextLine(input, inputState);

        if (lineBytes == null) {
            return StreamState.INPUT_NEEDED;
        }

        String lineString = StringUtils.newString(lineBytes);

        try {
            Map<String, Object> map = gson.fromJson(lineString, parseType);

            if (map == null) {
```

```
        continue;
    }

    output.setRowFromMap(map);
    // No overrides needed, so just call next() here.
    output.next();
} catch (Exception ex) {
    setReject(lineString, ex);
    return StreamState.REJECT;
}
}
```

The factory, `JsonParserFactory.java`, instantiates and returns a parser in the `prepare()` method. No additional setup is required.

Connecting to Vertica

This book explains several methods of connecting to Vertica, including:

- Directly connecting to Vertica using the vsql client application.
- Installing and configuring the Vertica client libraries to allow client applications to access Vertica.
- Developing your own client applications using the Vertica client libraries.

Using vsql

vsql is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.

If you are using the vsql client installed on the server, then you can connect from the:

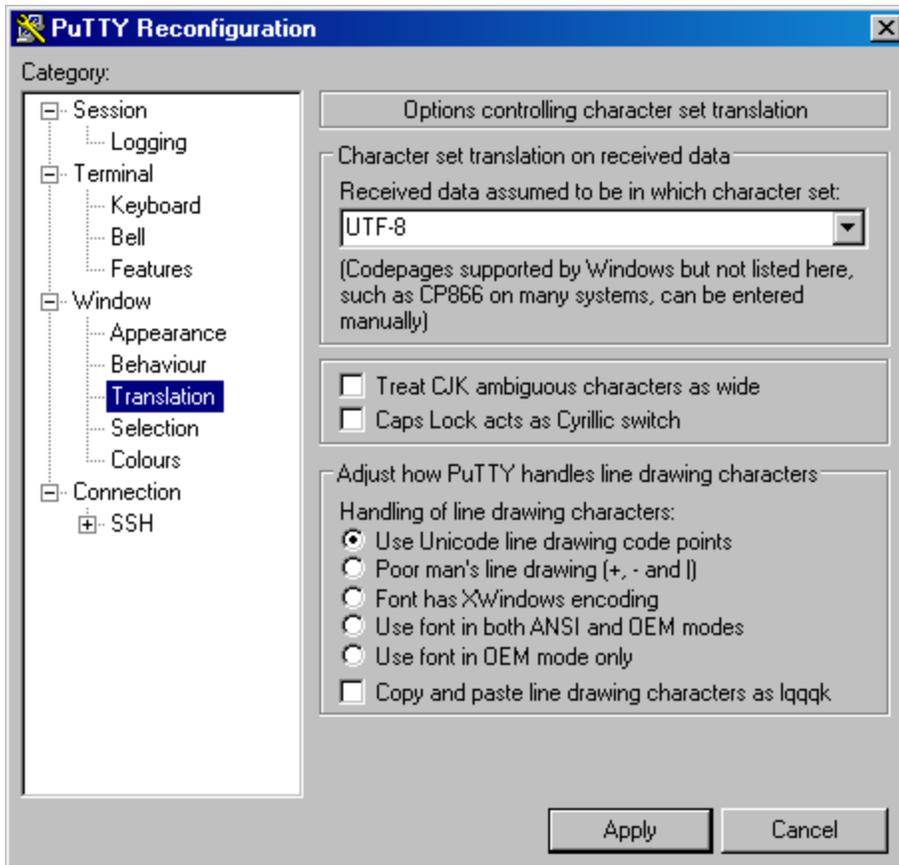
- [Administration Tools](#)
- [Linux command line](#)

You can also [install the vsql client](#) for other supported platforms.

A man page is available for vsql. If you are running as the dbadmin user, simply type: `man vsql`. If you are running as a different user, type: `man -M /opt/vertica/man vsql`.

General Notes

- SQL statements can be spread over several lines for clarity.
- vsql can handle input and output in UTF-8 encoding. The terminal emulator running vsql must be set up to display the UTF-8 characters correctly. Follow the documentation of your terminal emulator. The following example shows the settings in PuTTY from the Change Settings > Window > Translation option:



See also [Best Practices for Working with Locales](#).

- Cancel SQL statements by typing Ctrl+C.
- Traverse command history by typing Ctrl+R.
- When you disconnect a user session, any transactions in progress are automatically rolled back.
- To view wide result sets, use the Linux `less` utility to truncate long lines.
 - a. Before connecting to the database, specify that you want to use `less` for query output:

```
$ export PAGER=less
```

- b. Connect to the database.
- c. Query a wide table:

```
=> select * from wide_table;
```

d. At the `less` prompt, type:

```
-S
```

- If a shell running `vsq1` fails (crashes or freezes), the `vsq1` processes continue to run even if you stop the database. In that case, log in as `root` on the machine on which the shell was running and manually kill the `vsq1` process. For example:

```
# ps -ef | grep vertica
#
fred 2401 1 0 06:02 pts/1 00:00:00 /opt/vertica/bin/vsq1 -p 5433 -
h test01_site01 quick_start_single
#
# kill -9 2401
```

Installing the vsql Client

The vsql client is installed as part of the Vertica server rpm. It is also available as a download for other Unix-based systems such as HP-UX, AIX, and Mac OSX.

Unix Installation

1. Use a web browser to log in to the [myVertica portal](#).
2. Click Downloads, and choose Client Drivers.
3. Download the appropriate vsql client. There are both 32-bit and 64-bit versions for most platforms.

Important: Vertica provides the FIPS-compliant client driver only as an rpm for 64-bit clients. You can install this rpm only on FIPS-enabled machines. The FIPS client includes vsql and ODBC drivers. If you are installing the FIPS-specific client, refer to the section, [Installing the FIPS Client Driver for ODBC and vsql](#).

4. Extract the tarball. The tarball is organized to extract into /opt/vertica if you extract it at the root (/) of the drive.
5. Optionally add the directory where the vsql client resides to your path.
6. Verify mode on the vsql file is executable. For example: `chmod ugo+x vsql_VERSION`
7. Set your shell locale to a locale supported by vsql. For example, in your .profile, add,
`export LANG=en_US.UTF-8`

Windows Installation

vsql on Windows is installed as part of the Windows Client Driver package. For installation details, see [The Vertica Client Drivers and Tools for Windows](#).

See [Using vsql for Windows Users](#) for details on using vsql in a Windows console.

Connecting From the Administration Tools

You can use the Administration Tools to connect to a database using vsql on any node in the cluster.

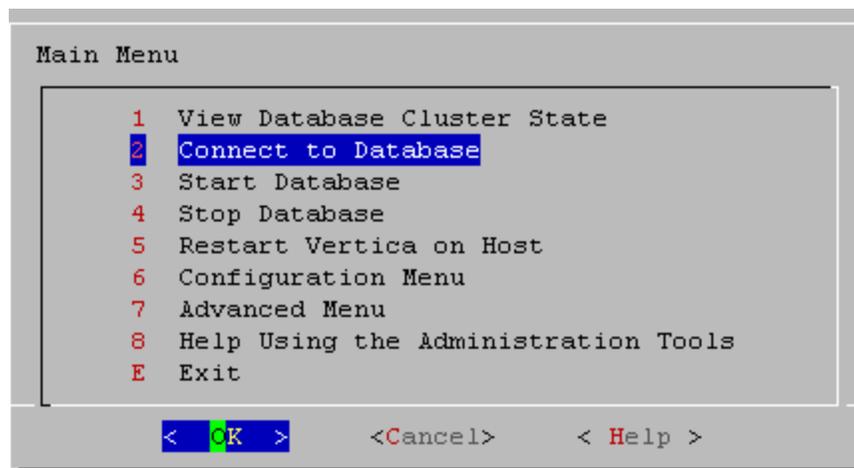
1. Log in as the database administrator user; for example, dbadmin.

Note: Vertica does not allow users with root privileges to connect to a database for security reasons.

2. Run the Administration Tools.

```
/opt/vertica/bin/admintools
```

3. On the Main Menu, select **Connect to Database**.



4. Supply the database password if asked:

Password:

When you create a new user with the [CREATE USER](#) command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the [ALTER USER](#) command.

5. The Administration Tools connect to the database and transfer control to vsql.

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.  
Type: \h or \? for help with vsql commands  
      \g or terminate with semicolon to execute query  
      \q to quit  
  
=>
```

Note: See [Meta-Commands](#) for the various commands you can run while connected to the database through the Administration Tools.

Connecting from the Command Line

You can connect to a database using `vsql` from the command line on multiple client platforms.

If the connection cannot be made for any reason—for example, you have insufficient privileges, or the server is not running on the targeted host—`vsql` returns an error and terminates.

Syntax

```
/opt/vertica/bin/vsql [-h host] [ option...] [ dbname [ username ] ]
```

Parameters

<i>host</i>	Optional if you connect to a local server. You can provide an IPv4 or IPv6 IP address or a host name. For Vertica servers that have both IPv4 and IPv6 addressed and you have provided a host name instead of an IP address, you can prefer to use an IPv4 address with the <code>-4</code> option and to use the IPv6 address with the <code>-6</code> option if the DNS is configured to provide both IPv4 and IPv6 addresses. If you are using IPv6 and provide an IP address, you must append the address with an <code>%<i>interface name</i></code> .
<i>option</i>	One or more <code>vsql</code> Command-Line Options If the database is password protected, you must specify the <code>-w</code> or <code>--password</code> command line option.
<i>dbname</i>	The name of the target database, by default your Linux user name.
<i>username</i>	A database username, by default your Linux user name.

Exit Codes

`vsql` returns 0 to the shell when it terminates normally. Otherwise, it returns one of the following:

- 1: A fatal error occurred—for example, out of memory or file not found.
- 2: The connection to the server went bad and the session was not interactive
- 3: An error occurred in a script and the variable `ON_ERROR_STOP` was set.
- Unrecognized words in the command line might be interpreted as database or user names.

Examples

The following example shows how to capture error messages by redirecting `vsql` output to the output file `retail_queries.out`:

```
$ vsql --echo-all < retail_queries.sql > retail_queries.out 2>&1
```

Command-Line Options

This section contains the command-line options for `vsql`.

General Options

Option	Description
<code>--command <i>command</i></code> <code>-c <i>command</i></code>	Runs one command and exits. This command is useful in shell scripts.
<code>--dbname <i>dbname</i></code> <code>-d <i>dbname</i></code>	Specifies the name of the database to which you want to connect. Using this command is equivalent to specifying <i>dbname</i> as the first non-option argument on the command line.
<code>--file <i>filename</i></code> <code>-f <i>filename</i></code>	Uses the <i>filename</i> as the source of commands instead of reading commands interactively. After the file is processed, <code>vsql</code> terminates.
<code>--help</code>	Displays help about <code>vsql</code> command line arguments and exits.
<code>--timing</code> <code>-i</code>	Enables the <code>\timing</code> meta-command.

<code>--list</code> <code>-l</code>	Returns all available databases, then exits. Other non-connection options are ignored. This command is similar to the internal command <code>\list</code> .
<code>--set assignment</code> <code>--variable assignment</code> <code>-v assignment</code>	Performs a variable assignment, like the <code>\set</code> internal command.
<code>--version -V</code>	Prints the vsql version and exits.
<code>--no-vsqr</code> <code>-X</code>	Disables all command line editing and history functionality.

Connection Options

Option	Description
<code>-4</code>	When resolving hostnames in dual stack environments, prefer IPv4 addresses.
<code>-6</code>	When resolving hostnames in dual stack environments, prefer IPv6 addresses.
<code>-B server:port[,...]</code>	Sets connection backup server/port. Use comma-separated multiple hosts (default: not set). If using an IPv6 address, enclose the address in brackets ([,]) and place the port outside of the brackets. For example <code>\B [2620:0:a13:8a4:9d9f:e0e3:1181:7f51]:5433</code>
<code>--enable-connection-load-balance -C</code>	Enables connection load balancing (default: not enabled). Note: You can only use load balancing with one address family in dual stack environments. For example, if you've configured load balancing for IPv6 addresses, then when an IPv4 client connects and requests load balancing the server does not allow it.
<code>--host hostname</code> <code>-h hostname</code>	Specifies the host name of the machine on which the server is running.
<code>-k krb-service</code>	Provides the service name portion of the Kerberos principal (default: vertica). Using <code>-k</code> is equivalent to using the drivers'

	KerberosServiceName connection string.
<code>-K <i>krb-host</i></code>	Provides the instance or host name portion of the Kerberos principal. <code>-K</code> is equivalent to the drivers' KerberosHostName connection string.
<code>--sslmode</code> <code>-m</code>	Specifies the policy for making SSL connections to the server. Options are <code>require</code> , <code>prefer</code> , <code>allow</code> , and <code>disable</code> . You can also set the <code>VSQL_SSLMODE</code> variable to achieve the same effect. If the variable is set, the command-line option overrides it.
<code>--port <i>port</i></code> <code>-p <i>port</i></code>	Specifies the TCP port or the local socket file extension on which the server is listening for connections. Defaults to port 5433.
<code>--username <i>username</i></code> <code>-U <i>username</i></code>	Connects to the database as the user <i>username</i> instead of the default.
<code>-w <i>password</i></code>	Specifies the password for a database user. Note: Using this command-line option displays the database password in plain text. Use it with care, particularly if you are connecting as the database administrator, to avoid exposing sensitive information.
<code>--password</code> <code>-W</code>	Forces <code>vsql</code> to prompt for a password before connecting to a database. The password is not displayed on the screen. This option remains set for the entire session, even if you change the database connection with the meta-command <code>\connect</code> .

Output Formatting

Option	Description
<code>--no-align</code> <code>-A</code>	Switches to unaligned output mode. (The default output mode is aligned.)
<code>-b</code>	Beep on command completion.
<code>--field-separator</code> <code><i>separator</i></code>	Specifies the field separator for unaligned output (default: <code>" "</code>) (<code>-P fieldsep=</code>). (See <code>-A --no-align</code> .) Using this

<code>-F separator</code>	command is equivalent to <code>\pset fieldsep</code> or <code>\f</code> .
<code>--html</code> <code>-H</code>	Turns on HTML tabular output. Using this command is equivalent to using the <code>\pset format html</code> or the <code>\H</code> command.
<code>--pset assignment</code> <code>-P assignment</code>	Lets you specify printing options in the style of <code>\pset</code> on the command line. You must separate the name and value with an equals (=) sign instead of a space. Thus, to set the output format to LaTeX, you could write <code>-P format=latex</code> .
<code>-Q</code>	Turns on trailing record separator. Use <code>\pset trailingrecordsep</code> to toggle the trailing record separator on or off.
<code>--record-separator separator</code> <code>-R separator</code>	Uses <i>separator</i> as the record separator. Using this command is equivalent to using the <code>\pset recordsep</code> command.
<code>--tuples-only</code> <code>-t</code>	Disables printing of column names, result row count footers, and so on. This is equivalent to the <code>\t</code> command.
<code>--table-attr options</code> <code>-T options</code>	Allows you to specify options to be placed within the HTML table tag. See <code>\pset</code> for details.
<code>--extended</code> <code>-x</code>	Enables extended table formatting mode. This is equivalent to the command <code>\x</code> .

Input and Output Options

Option	Description
<code>--echo-call</code> <code>-a</code>	Prints all input lines to standard output as they are read. This approach is more useful for script processing than interactive mode. It is the same as setting the variable ECHO to <code>all</code> .
<code>--echo-queries</code> <code>-e</code>	Copies all SQL commands sent to the server to standard output. Using this command is equivalent to setting the variable <code>ECHO</code> to <code>queries</code> .

-E	Displays queries generated by internal commands.
-n	Disables command line editing.
--output <i>filename</i> -o <i>filename</i>	Writes all query output into file <i>filename</i> . Using this command is equivalent to using the command \o.
--quiet -q	Specifies that vsql do its work quietly (without informational output, such as welcome messages). This command is useful with the -c option. Within vsql you can also set the QUIET variable to achieve the same effect.
--single-step -s	Runs in single-step mode for debugging scripts. Forces vsql to prompt before each statement is sent to the database and allows you to cancel execution.
--single-line -S	Runs in single-line mode where a newline terminates a SQL command, as if you are using a semicolon. Note: This mode is provided only by customer request. Micro Focus recommends that you not use single-line mode in cases where you mix SQL and meta-commands on a line. In single-line mode, the order of execution might be unclear to the inexperienced user.

-c Command --command Command

-c command --command command runs one command and exits. This is useful in shell scripts.

Use either:

- A command string that can be completely parsed by the server that does not contain features specific to vsql
- A single meta-command

You cannot mix SQL and vsql meta-commands. You can, however, pipe the string into vsql as shown:

```
echo "\\timing\\select * from t" | ../Linux64/bin/vsql
Timing is on.
i | c | v
---+---
(0 rows)
```

Note: If you use double quotes (") with echo, you must double the backslashes (\).

-f Filename --file Filename

`-f filename --file filename` uses *filename* as the source of commands instead of reading commands interactively. After the file is processed, `vsq1` terminates.

If *filename* is a hyphen (-), the standard input is read.

Using this option is different from writing `vsq1 < filename`. Using `-f` enables some additional features such as error messages with line numbers. Conversely, the variant using the shell's input redirection should always yield exactly the same output that you would have gotten had you entered everything manually.

-F Separator --field-separator Separator

`-F separator --field-separator separator` specifies the field separator for unaligned output (default: "|") (`-P fieldsep=`). (See `-A --no-align`.) This is equivalent to `\pset fieldsep` or `\f`.

To set the field separator value to a control character, use your shell's control character escape notation. In Bash, you specify a control character in an argument using a dollar sign (\$) followed by a string contained in single quotes. This string can contain C-string escapes (such as `\t` for tab), or a backslash (\) followed by an octal value for the character you want to use.

The following example demonstrates setting the separator character to tab (`\t`), vertical tab (`\v`) and the octal value of vertical tab (`\013`).

```
$ vsq1 -At -c "SELECT * FROM testtable;"
A|1|2|3
B|4|5|6

$ vsq1 -F $'\t' -At -c "SELECT * FROM testtable;"
A      1      2      3
B      4      5      6

$ vsq1 -F $'\v' -At -c "SELECT * FROM testtable;"
A
 1
 2
 3
B
```

```
4
5
6
$ vsql -F '$'\013' -At -c "SELECT * FROM testtable;"
A
1
2
3
B
4
5
6
```

-h Hostname --host Hostname

-h hostname --host hostname specifies the host name of the machine on which the server is running.

Notes:

- If you are using client authentication with a Kerberos connection method of either gss or krb5, you are required to specify -h *hostname*.
- If you are using client authentication with a "local" connection type specified, and you want to match the client authentication entry, do not use -h hostname.

-i -- timing

The command line option -i enables the \timing meta-command. You can only use this command with the -c [Command](#) --command [Command](#) and -f [Filename](#) --file [Filename](#) commands:

```
$VSQL -h host1 -U user1 -d VMart -p 15 -w ***** -i -f transactions.sql
```

For more information see [\timing](#).

-v Assignment --set Assignment --variable Assignment

-v assignment --set assignment --variable assignment performs a variable assignment, like the \set internal command.

Note: You must separate name and value, if any, by an equals sign (=) on the command line.

To unset a variable, omit the equal sign. To set a variable without a value, use the equals sign but omit the value. Make these assignments at a very early stage of start-up, so that variables reserved for internal purposes can get overwritten later.

Connecting From a Non-Cluster Host

You can use the Vertica vsql executable image on a non-cluster Linux host to connect to a Vertica database.

- On Red Hat, CentOS, and SUSE systems, you can install the client driver RPM, which includes the vsql executable. See [Installing the Client RPM on Red Hat and SUSE](#) for details.
- If the non-cluster host is running the same version of Linux as the cluster, copy the image file to the remote system. For example:

```
$ scp host01:/opt/vertica/bin/vsql . $ ./vsql
```

- If the non-cluster host is running a different version of Linux than your cluster hosts, and that operating system is not Red Hat version 5 64-bit or SUSE 10/11 64-bit, you must install the Vertica server RPM in order to get vsql. Download the appropriate rpm package from the Download tab of the [myVertica portal](#) then log into the non-cluster host as root and install the rpm package using the command:

```
# rpm -Uvh filename
```

In the above command, *filename* is the package you downloaded. Note that you do not have to run the `install_Vertica` script on the non-cluster host in order to use vsql.

Notes

- Use the same [Command-Line Options](#) that you would on a cluster host.
- You cannot run vsql on a Cygwin bash shell (Windows). Use ssh to connect to a cluster host, then run vsql.

Meta-Commands

Anything you enter in vsql that begins with an unquoted backslash is a vsql meta-command that is processed by vsql itself. These commands help make vsql more useful for administration or scripting. Meta-commands are more commonly called slash or backslash commands.

The format of a vsql command is the backslash, followed immediately by a command verb, then any arguments. The arguments are separated from the command verb and each other by any number of whitespace characters.

To include whitespace into an argument you can quote it with a single quote. To include a single quote into such an argument, precede it by a backslash. Anything contained in single quotes is furthermore subject to C-like substitutions for `\n` (new line), `\t` (tab), `\digits`, `\0digits`, and `\0xdigits` (the character with the given decimal, octal, or hexadecimal code).

If an unquoted argument begins with a colon (:), it is taken as a vsql variable and the value of the variable is used as the argument instead.

Arguments that are enclosed in backquotes (```) are taken as a command line that is passed to the shell. The output of the command (with any trailing newline removed) is taken as the argument value. The above escape sequences also apply in backquotes.

Some commands take a SQL identifier (such as a table name) as argument. These arguments follow the syntax rules of SQL: Unquoted letters are forced to lowercase, while double quotes (`"`) protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotes, paired double quotes reduce to a single double quote in the resulting name. For example, `FOO"BAR"BAZ` is interpreted as `fooBARbaz`, and `"A weird" name"` becomes `A weird" name`.

Parsing for arguments stops when another unquoted backslash occurs. This is taken as the beginning of a new meta-command. The special sequence `\\` (two backslashes) marks the end of arguments and continues parsing SQL commands, if any. That way SQL and vsql commands can be freely mixed on a line. But in any case, the arguments of a meta-command cannot continue beyond the end of the line.

`\! [COMMAND]`

`\! [COMMAND]` executes a command in a Linux shell (passing arguments as entered) or starts an interactive shell.

\?

\? displays help information about the meta-commands. Works the same as \h .

```
=> \?
General
  \c[onnect] [DBNAME]- [USER]]
           connect to new database (currently "VMart")
  \cd [DIR]
           change the current working directory
  \q
           quit vsql
  \set [NAME [VALUE]]
           set internal variable, or list all if no parameters
  \timing
           toggle timing of commands (currently off)
  \unset NAME
           unset (delete) internal variable
  \! [COMMAND]
           execute command in shell or start interactive shell
  \password [USER]
           change user's password

Query Buffer
  \e [FILE]
           edit the query buffer (or file) with external editor
  \g
           send query buffer to server
  \g FILE
           send query buffer to server and results to file
  \g | COMMAND
           send query buffer to server and pipe results to command
  \p
           show the contents of the query buffer
  \r
           reset (clear) the query buffer
  \s [FILE]
           display history or save it to file
  \w FILE
           write query buffer to file

Input/Output
  \echo [STRING]
           write string to standard output
  \i FILE
           execute commands from file
  \o FILE
           send all query results to file
  \o | COMMAND
           pipe all query results to command
  \o
           close query-results file or pipe
  \qecho [STRING]
           write string to query output stream (see \o)

Informational
  \d [PATTERN]
           describe tables (list tables if no argument is supplied)
  \df [PATTERN]
           list functions
  \dj [PATTERN]
           list projections
  \dn [PATTERN]
           list schemas
  \dp [PATTERN]
           list table access privileges
  \ds [PATTERN]
           list sequences
  \dS [PATTERN]
           list system tables
  \dt [PATTERN]
           list tables
  \dtv [PATTERN]
           list tables and views
  \dT [PATTERN]
           list data types
  \du [PATTERN]
           list users
  \dv [PATTERN]
           list views
  \l
           list all databases
  \z [PATTERN]
           list table access privileges (same as \dp)

Formatting
  \a
           toggle between unaligned and aligned output mode
  \b
           toggle beep on command completion
  \C [STRING]
           set table title, or unset if none
  \f [STRING]
           show or set field separator for unaligned query output
  \H
           toggle HTML output mode (currently off)
  \pset NAME [VALUE]
```

```
set table output option  
(NAME := {format|border|expanded|fieldsep|footer|null|  
recordsep|tuples_only|title|tableattr|pager})  
\t show only rows (currently off)  
\T [STRING] set HTML <table> tag attributes, or unset if none  
\x toggle expanded output (currently off)
```

\a

\a toggles output format alignment. This command is kept for backwards compatibility. See [\pset](#) for a more general solution.

\a is similar to the command line option `-A --no-align`, which only disables alignment.

\b

\b toggles beep on command completion.

\c (or \connect) [Dbname [Username]]

\c (or \connect) [dbname [username]] establishes a connection to a new database and/or under a user name. The previous connection is closed. If dbname is - the current database name is assumed.

If username is omitted the current user name is assumed.

As a special rule, \connect without any arguments connects to the default database as the default user (as you would have gotten by starting vsql without any arguments).

If the connection attempt fails (wrong user name, access denied, or other error messages), the previous connection is kept if and only if vsql is in interactive mode. When executing a non-interactive script, processing immediately stops with an error. This is a distinction that avoids typos and prevents scripts from accidentally acting on the wrong database.

\C [STRING]

\C [STRING] sets the title of any tables being printed as the result of a query or unsets any such title. This command is equivalent to `\pset title title`. (The name of this command derives from "caption," as it was previously only used to set the caption in an HTML table.)

\cd [DIR]

\cd [DIR] changes the current working directory to *directory*. Without argument, changes to the current user's home directory.

To print your current working directory, use \! pwd. For example:

```
=> \!pwd
/home/dbadmin
```

The \d [PATTERN] Meta-Commands

This section describes the various \d meta-commands.

All \d meta-commands take an optional pattern (asterisk [*] or question mark [?]) and return only the records that match that pattern.

The ? argument is useful if you can't remember if a table name uses an underscore or a dash:

```
=> \dn v?internal
List of schemas
  Name   | Owner
-----+-----
v_internal | dbadmin
(1 row)
```

The output from the \d metacommands places double quotes around non-alphanumeric table names and table names that are keywords, such as in the following example.

```
=> CREATE TABLE my_keywords.precision(x numeric (4,2));
CREATE TABLE
=> \d
List of tables
  Schema   | Name           | Kind | Owner
-----+-----+-----+-----
my_keywords | "precision"   | table | dbadmin
```

Double quotes are optional when you use a \d command with pattern matching.

\d [PATTERN]

The \d meta-command lists all tables in the database and returns their schema, table name, kind (e.g., table), and owner.

If you use `\d [PATTERN]` and provide the schema name or table name (or wildcard or ? characters) as the pattern argument, the result shows more detailed information about the tables:

- Schema name
- Table name
- Column name
- Column data type
- Data type size
- Default column value
- Whether the column accepts null values or has a NOT NULL constraint
- Whether there is a primary key or foreign key constraint

To view information about system tables, you must include the `V_MONITOR` or `V_CATALOG` as the pattern argument; for example:

```
\d v_catalog.types -- information on the types table in v_catalog schema
\d v_catalog.*    -- information on all table columns in v_catalog schema
```

The following output is the result of all tables in the [vmart schema](#), which is in the PUBLIC schema.

```
VMart=> \d
          List of tables
  Schema |          Name          | Kind | Owner
-----+-----+-----+-----
online_sales | call_center_dimension | table | dbadmin
online_sales | online_page_dimension | table | dbadmin
online_sales | online_sales_fact     | table | dbadmin
public      | customer_dimension    | table | dbadmin
public      | date_dimension        | table | dbadmin
public      | employee_dimension    | table | dbadmin
public      | inventory_fact        | table | dbadmin
public      | product_dimension     | table | dbadmin
public      | promotion_dimension   | table | dbadmin
public      | shipping_dimension    | table | dbadmin
public      | vendor_dimension      | table | dbadmin
public      | warehouse_dimension   | table | dbadmin
store       | store_dimension       | table | dbadmin
store       | store_orders_fact     | table | dbadmin
store       | store_sales_fact      | table | dbadmin
(15 rows)
```

This example returns information on the `inventory_fact` table in the VMart database:

```
VMart=> \x
Expanded display is on.
VMart=> \d inventory_fact
List of Fields by Tables
-[ RECORD 1 ]-----
Schema      | public
Table       | inventory_fact
Column      | date_key
Type        | int
Size        | 8
Default     |
Not Null    | t
Primary Key | f
Foreign Key | public.date_dimension(date_key)
-[ RECORD 2 ]-----
Schema      | public
Table       | inventory_fact
Column      | product_key
Type        | int
Size        | 8
Default     |
Not Null    | t
Primary Key | f
Foreign Key | public.product_dimension(product_key)
-[ RECORD 3 ]-----
Schema      | public
Table       | inventory_fact
Column      | product_version
Type        | int
Size        | 8
Default     |
Not Null    | t
Primary Key | f
Foreign Key | public.product_dimension(product_version)
-[ RECORD 4 ]-----
Schema      | public
Table       | inventory_fact
Column      | warehouse_key
Type        | int
Size        | 8
Default     |
Not Null    | t
Primary Key | f
Foreign Key | public.warehouse_dimension(warehouse_key)
-[ RECORD 5 ]-----
Schema      | public
Table       | inventory_fact
Column      | qty_in_stock
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
```

Use the question mark [?] argument to replace a single character. For example, the ? argument replaces the last character in the user-created SubQ1 and SubQ2 tables, so the command returns information about both:

```
=> \d SubQ?
```

List of Fields by Tables

Schema	Table	Column	Type	Size	Default	Not Null	Primary Key	Foreign Key
public	SubQ1	a	int	8		f	f	
public	SubQ1	b	int	8		f	f	
public	SubQ1	c	int	8		f	f	
public	SubQ2	x	int	8		f	f	
public	SubQ2	y	int	8		f	f	
public	SubQ2	z	int	8		f	f	

(6 rows)

If you run the `\d` command and provide both the schema and table name, output includes columns for tables that match the pattern

```
VMart=> \x
Expanded display is on.
VMart=> \d v_catalog.types
List of Fields by Tables
-[ RECORD 1 ]-----
Schema      | v_catalog
Table       | types
Column      | column_size
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 2 ]-----
Schema      | v_catalog
Table       | types
Column      | creation_parameters
Type        | varchar(128)
Size        | 128
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 3 ]-----
Schema      | v_catalog
Table       | types
Column      | epoch
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 4 ]-----
Schema      | v_catalog
Table       | types
Column      | interval_mask
Type        | int
Size        | 8
Default     |
Not Null    | f
```

```
Primary Key | f
Foreign Key |
-[ RECORD 5 ]-----
Schema      | v_catalog
Table       | types
Column      | max_scale
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 6 ]-----
Schema      | v_catalog
Table       | types
Column      | min_scale
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 7 ]-----
Schema      | v_catalog
Table       | types
Column      | odbc_subtype
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 8 ]-----
Schema      | v_catalog
Table       | types
Column      | odbc_type
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 9 ]-----
Schema      | v_catalog
Table       | types
Column      | type_id
Type        | int
Size        | 8
Default     |
Not Null    | f
Primary Key | f
Foreign Key |
-[ RECORD 10 ]-----
Schema      | v_catalog
Table       | types
Column      | type_name
Type        | varchar(128)
Size        | 128
Default     |
Not Null    | f
```

```
Primary Key | f  
Foreign Key |
```

To view all tables in a schema, use the wildcard character. The following command would return all system tables in the V_CATALOG schema:

```
=> \d v_catalog.*
```

\df [PATTERN]

The `\df [PATTERN]` meta-command returns all function names, the function return data type, and the function argument data type. Also returns the procedure names and arguments for all procedures that are available to the user.

```
vmartdb=> \df  
  
                List of functions  
procedure_name | procedure_return_type | procedure_argument_types  
-----+-----+-----  
abs            | Float                 | Float  
abs            | Integer               | Integer  
abs            | Interval               | Interval  
abs            | Interval               | Interval  
abs            | Numeric                | Numeric  
acos           | Float                 | Float  
...  
width_bucket  | Integer                | Float, Float, Float, Integer  
width_bucket  | Integer                | Interval, Interval, Interval, Integer  
width_bucket  | Integer                | Interval, Interval, Interval, Integer  
width_bucket  | Integer                | Timestamp, Timestamp, Timestamp, Integer  
...
```

The following example uses the wildcard character to search for all functions that begin with `as`:

```
vmartdb=> \df as*  
  
                List of functions  
procedure_name | procedure_return_type | procedure_argument_types  
-----+-----+-----  
ascii          | Integer                | Varchar  
asin           | Float                  | Float  
(2 rows)
```

\dj [PATTERN]

The `\dj [PATTERN]` meta-command returns all projections showing the schema, projection name, owner, and node. The returned rows include superprojections, live aggregate projections, Top-K projections, and projections with expressions:

```
vmartdb=> \dj
                List of projections
  Schema | Name | Owner | Node
-----+-----+-----+-----
public  | product_dimension_node0001 | dbadmin | v_wmartdb_node0001
public  | product_dimension_node0002 | dbadmin | v_wmartdb_node0002
public  | product_dimension_node0003 | dbadmin | v_wmartdb_node0003
online_sales | call_center_dimension_node0001 | dbadmin | v_wmartdb_node0001
online_sales | call_center_dimension_node0002 | dbadmin | v_wmartdb_node0002
online_sales | call_center_dimension_node0003 | dbadmin | v_wmartdb_node0003
...
```

If you supply a projection name as an argument, the system returns fewer records:

```
vmartdb=> \dj call_center_dimension_n*
                List of projections
  Schema | Name | Owner | Node
-----+-----+-----+-----
online_sales | call_center_dimension_node0001 | dbadmin | v_wmartdb_node0001
online_sales | call_center_dimension_node0002 | dbadmin | v_wmartdb_node0002
online_sales | call_center_dimension_node0003 | dbadmin | v_wmartdb_node0003
(3 rows)
```

\dn [PATTERN]

The `\dn [PATTERN]` meta-command returns the schema names and schema owner.

```
vmartdb=> \dn
  List of schemas
  Name | Owner
-----+-----
v_internal | dbadmin
v_catalog | dbadmin
v_monitor | dbadmin
public | dbadmin
store | dbadmin
online_sales | dbadmin
(6 rows)
```

The following command returns all schemas that begin with the letter v:

```
=> \dn v*
  List of schemas
  Name | Owner
-----+-----
v_internal | dbadmin
v_catalog | dbadmin
v_monitor | dbadmin
(3 rows)
```

\dp [PATTERN]

The \dp [PATTERN] meta-command returns the grantee, grantor, privileges, schema, and name for all table access privileges in each schema:

```
vmartdb=> \dp
      Access privileges for database "vmartdb"
  Grantee | Grantor | Privileges | Schema | Name
-----+-----+-----+-----+-----
          | dbadmin | USAGE     |        | public
          | dbadmin | USAGE     |        | v_internal
          | dbadmin | USAGE     |        | v_catalog
          | dbadmin | USAGE     |        | v_monitor
(4 rows)
```

Note: \dp is the same as [\z](#).

\ds [PATTERN]

The \ds [PATTERN] meta-command (lowercase s) returns a list of sequences and their parameters.

The following series of commands creates a sequence called my_seq and uses the vsql command to display its parameters:

```
=> CREATE SEQUENCE my_seq MAXVALUE 5000 START 150;
CREATE SEQUENCE

=> \ds
                        List of Sequences
  Schema | Sequence | CurrentValue | IncrementBy | Minimum | Maximum | AllowCycle
-----+-----+-----+-----+-----+-----+-----
public | my_seq   |          149 |           1 |         1 |      5000 | f
(1 row)
```

Note: You can return additional information about sequences by issuing `SELECT * FROM SEQUENCES`, as described in the SQL Reference Manual.

\dS [PATTERN]

The \dS [PATTERN] meta-command (uppercase S) returns all system table (monitoring API) names from the [V_CATALOG](#) and [V_MONITOR](#) schemas.

Tip: You can get identical results running this query: `SELECT * FROM system_tables;`

If you specify a schema name, you can view results for tables in that schema only; however, you must use the wildcard character. For example:

```
=> \dS v_catalog.*
```

You can also run the `\dS` command with a table argument to return information for that table only:

```
vmartdb=> \dS columns
                List of tables
 Schema | Name | Kind | Description | Comment
-----+-----+-----+-----+-----
v_catalog | columns | system | Table column information |
(1 row)
```

If you provide the schema name with the table name, the output returns information about the table:

```
vmartdb=> \dS v_catalog.types
                List of tables
 Schema | Name | Kind | Description | Comment
-----+-----+-----+-----+-----
v_catalog | types | system | Information about supported data types |
(1 row)
```

`\dt [PATTERN]`

The `\dt [PATTERN]` meta-command (lowercase t) is identical to `\d` and returns all tables in the database—unless a table name is specified—in which case the command lists only the schema, name, kind and owner for the specified table (or tables if wildcards used).

```
vmartdb=> \dt inventory_fact
                List of tables
 Schema | Name | Kind | Owner
-----+-----+-----+-----
public | inventory_fact | table | dbadmin
(1 row)
```

The following command returns all table names that begin with "st":

```
vmartdb=> \dt st*
                List of tables
 Schema | Name | Kind | Owner
-----+-----+-----+-----
store | store_dimension | table | dbadmin
store | store_orders_fact | table | dbadmin
store | store_sales_fact | table | dbadmin
(3 rows)
```

\dT [PATTERN]

The \dT [PATTERN] meta-command (uppercase T) lists all supported data types.

```
vmartdb=> \dT
List of data types
  type_name
-----
Binary
Boolean
Char
Date
Float
Integer
Interval Day
Interval Day to Hour
Interval Day to Minute
Interval Day to Second
Interval Hour
Interval Hour to Minute
Interval Hour to Second
Interval Minute
Interval Minute to Second
Interval Month
Interval Second
Interval Year
Interval Year to Month
Numeric
Time
TimeTz
Timestamp
TimestampTz
Varbinary
Varchar
(26 rows)
```

\dtv [PATTERN]

The \dtv [PATTERN] meta-command lists all tables and views, returning the schema, table or view name, kind (table of view), and owner.

```
vmartdb=> \dtv
          List of tables
 Schema | Name | Kind | Owner
-----+-----+-----+-----
online_sales | call_center_dimension | table | release
online_sales | online_page_dimension | table | release
online_sales | online_sales_fact | table | release
public | customer_dimension | table | release
public | date_dimension | table | release
public | employee_dimension | table | release
public | inventory_fact | table | release
```

```

public      | product_dimension | table | release
public      | promotion_dimension | table | release
public      | shipping_dimension | table | release
public      | vendor_dimension | table | release
public      | warehouse_dimension | table | release
store       | store_dimension | table | release
store       | store_orders_fact | table | release
store       | store_sales_fact | table | release
(15 rows)

```

\du [PATTERN]

The `\du [PATTERN]` meta-command returns all database users and attributes, such as if user is a superuser.

```

vmartdb=> \du
      List of users
User name | Is Superuser
-----+-----
dbadmin   | t
(1 row)

```

\dv [PATTERN]

The `\dv [PATTERN]` meta-command returns the schema name, view name, and view owner.

The following example defines a view using the SEQUENCES system table:

```

vmartdb=> CREATE VIEW my_seqview AS (SELECT * FROM sequences);
CREATE VIEW

vmartdb=> \dv
      List of views
Schema | Name | Owner
-----+-----+-----
public | my_seqview | dbadmin
(1 row)

```

If a view name is provided as an argument, the result shows the schema, view name, and the following for all columns within the view's result set: schema name, view name, column name, column data type, and data type size.

```

vmartdb=> \dv my_seqview
      List of View Fields
Schema | View | Column | Type | Size
-----+-----+-----+-----+-----
public | my_seqview | sequence_schema | varchar(128) | 128
public | my_seqview | sequence_name | varchar(128) | 128

```

```
public | my_seqview | owner_name      | varchar(128) | 128
public | my_seqview | identity_table_name | varchar(128) | 128
public | my_seqview | session_cache_count | int          | 8
public | my_seqview | allow_cycle        | boolean      | 1
public | my_seqview | output_ordered     | boolean      | 1
public | my_seqview | increment_by       | int          | 8
public | my_seqview | minimum            | int          | 8
public | my_seqview | maximum            | int          | 8
public | my_seqview | current_value      | int          | 8
public | my_seqview | sequence_schema_id | int          | 8
public | my_seqview | sequence_id        | int          | 8
public | my_seqview | owner_id           | int          | 8
public | my_seqview | identity_table_id  | int          | 8
(15 rows)
```

`\e \edit [FILE]`

`\e \edit [FILE]` edits the query buffer (or specified file) with an external editor. When the editor exits, its content is copied back to the query buffer. If no argument is given, the current query buffer is copied to a temporary file which is then edited in the same fashion.

The new query buffer is then re-parsed according to the normal rules of `vsq`, where the whole buffer up to the first semicolon is treated as a single line. (Thus you cannot make scripts this way. Use `\i` for that.) If there is no semicolon, `vsq` waits for one to be entered (it does not execute the query buffer).

Tip: `vsq` searches the environment variables `VSQ_EDITOR`, `EDITOR`, and `VISUAL` (in that order) for an editor to use. If all of them are unset, `vi` is used on Linux systems, `notepad.exe` on Windows systems.

`\echo [STRING]`

`\echo [STRING]` writes the string to standard output.

Tip: If you use the `\o` command to redirect your query output you might want to use `\qecho` instead of this command.

`\f [String]`

`\f [string]` sets the field separator for unaligned query output. The default is the vertical bar (`|`). See also `\pset` for a generic way of setting output options.

\g

The `\g` meta-command sends the query in the input buffer (see [\p](#)) to the server. With no arguments, it displays the results in the standard way.

`\g FILE` sends the query input buffer to the server, and writes the results to FILE.

`\g | COMMAND` sends the query buffer to the server, and pipes the results to a shell COMMAND.

See Also

- [\o](#)

\H

`\H` toggles HTML query output format. This command is for compatibility and convenience, but see `\pset` about setting other output options.

\h \help

`\h \help` displays help information about the meta-commands. Works the same as [\?](#).

```
=> \h
General
  \c[onnect] [DBNAME|- [USER]]
                        connect to new database (currently "VMart")
  \cd [DIR]             change the current working directory
  \q                   quit vsql
  \set [NAME [VALUE]]
                        set internal variable, or list all if no parameters
  \timing               toggle timing of commands (currently off)
  \unset NAME          unset (delete) internal variable
  \! [COMMAND]         execute command in shell or start interactive shell
  \password [USER]    change user's password

Query Buffer
  \e [FILE]            edit the query buffer (or file) with external editor
  \g                   send query buffer to server
  \g FILE              send query buffer to server and results to file
  \g | COMMAND         send query buffer to server and pipe results to command
  \p                   show the contents of the query buffer
  \r                   reset (clear) the query buffer
  \s [FILE]            display history or save it to file
```

```
\w FILE          write query buffer to file
Input/Output
\echo [STRING]   write string to standard output
\i FILE          execute commands from file
\o FILE          send all query results to file
\o | COMMAND     pipe all query results to command
\o              close query-results file or pipe
\qecho [STRING] write string to query output stream (see \o)
Informational
\d [PATTERN]     describe tables (list tables if no argument is supplied)
\df [PATTERN]   list functions
\dj [PATTERN]   list projections
\dn [PATTERN]   list schemas
\dp [PATTERN]   list table access privileges
\ds [PATTERN]   list sequences
\dS [PATTERN]   list system tables
\dT [PATTERN]   list tables
\dTV [PATTERN]  list tables and views
\dT [PATTERN]   list data types
\du [PATTERN]   list users
\dv [PATTERN]   list views
\l              list all databases
\z [PATTERN]    list table access privileges (same as \dp)
Formatting
\a              toggle between unaligned and aligned output mode
\b              toggle beep on command completion
\C [STRING]     set table title, or unset if none
\f [STRING]     show or set field separator for unaligned query output
\H              toggle HTML output mode (currently off)
\pset NAME [VALUE]
                set table output option
                (NAME := {format|border|expanded|fieldsep|footer|null|
                recordsep|tuples_only|title|tableattr|pager})
\t              show only rows (currently off)
\T [STRING]     set HTML <table> tag attributes, or unset if none
\x              toggle expanded output (currently off)
```

\i FILE

`\i filename` command reads input from the file *filename* and executes it as though it had been typed on the keyboard.

Note: To see the lines on the screen as they are read, set the variable `ECHO` to all.

Tip: The Vertica vsql client on Linux supports backquote (backtick) expansion. A simple example follows.

1. Set an environment variable to a path that contains scripts you want to run.

```
$ export MYSCRIPTS=/home/dbadmin/testscripts
```

2. Issue the vsql command.

```
$ vsql
```

3. Use backquote expansion to include the path for running an existing script (for example, sample.sql).

```
=> \i `echo $MYSCRIPTS/sample.sql`
```

\l

`\l` provides a list of databases and their owners.

```
vmartdb=> \l
  List of databases
  name  | user_name
-----+-----
vmartdb | dbadmin
(1 row)
```

Locale

The `vsql \locale` command displays the current locale setting or lets you set a new locale for the session.

This command does not alter the default locale for all database sessions. To change the default for all sessions, set the `DefaultSessionLocale` [configuration parameter](#).

Viewing the Current Locale Setting

To view the current locale setting, use the `vsql` command `\locale`, as follows:

```
=> \locale
en_US@collation=binary
```

Overriding the Default Local for a Session

To override the default local for a specific session, use the `vsql` command `\locale <ICU-locale-identifier>`. The session locale setting applies to any subsequent commands issued in the session.

For example:

```
\locale en_GBINFO:  
INFO 2567: Canonical locale: 'en_GBINFO:'  
Standard collation: 'LEN'  
English (GBINFO:)
```

Notes

The server locale settings impact only the collation behavior for server-side query processing. The client application is responsible for ensuring that the correct locale is set in order to display the characters correctly. Below are the best practices recommended by Micro Focus to ensure predictable results:

- The locale setting in the terminal emulator for vsql (POSIX) should be set to be equivalent to session locale setting on server side (ICU) so data is collated correctly on the server and displayed correctly on the client.
- The vsql locale should be set using the POSIX LANG environment variable in terminal emulator. Refer to the documentation of your terminal emulator for how to set locale.
- Server session locale should be set using the set as described in [Specify the Default Locale for the Database](#).
- Note that all input data for vsql should be in UTF-8 and all output data is encoded in UTF-8.
- Non UTF-8 encodings and associated locale values are not supported.

\o

The \o meta-command is used to control where vsql directs its query output. The output can be written to a file, piped to a shell command, or sent to the standard output.

\o FILE sends all subsequent query output to FILE.

\o | COMMAND pipes all subsequent query output to a shell COMMAND.

\o with no argument closes any open file or pipe, and switches back to normal query result output.

Notes

- Query results includes all tables, command responses, and notices obtained from the database server.
- To intersperse text output with query results, use `\qecho`.

See Also

- `\g`

`\p`

`\p` prints the current query buffer to the standard output. For example:

```
=> \p
CREATE VIEW my_seqview AS (SELECT * FROM sequences);
```

`\password [USER]`

`\password` starts the password change process. Users can only change their own passwords. The command prompts the user for the old password, a new password, and then the new password again to confirm.

A superuser can change the password of another user by supplying the username. A superuser is not prompted for the old password, either when changing his or her own password, or when changing another user's password.

Note: If you want to cancel the password change process, press ENTER until you return the to vsql prompt.

`\pset NAME [VALUE]`

`\pset NAME [VALUE]` sets options affecting the output of query result tables. NAME describes which option to set, as illustrated in the following table. The parameters of VALUE depend thereon.

It is an error to call `\pset` without arguments.

Adjustable printing options are:

format	<p>Sets the output format to one of <code>unaligned</code>, <code>aligned</code>, <code>html</code>, or <code>latex</code>. Unique abbreviations are allowed. (That would mean one letter is enough.)</p> <p>"Unaligned" writes all columns of a row on a line, separated by the currently active field separator. This is intended to create output that might be intended to be read in by other programs (tab-separated, comma-separated). "Aligned" mode is the standard, human-readable, nicely formatted text output that is default. The "HTML" and "LaTeX" modes put out tables that are intended to be included in documents using the respective mark-up language. They are not complete documents! (This might not be so dramatic in HTML, but in LaTeX you must have a complete document wrapper.)</p>
border	<p>The second argument must be a number. In general, the higher the number the more borders and lines the tables have, but this depends on the particular format. In HTML mode, this translates directly into the <code>border=...</code> attribute, in the others only values 0 (no border), 1 (internal dividing lines), and 2 (table frame) make sense.</p>
expanded	<p>Toggles between regular and expanded format. When expanded format is enabled, all output has two columns with the column name on the left and the data on the right. This mode is useful if the data wouldn't fit on the screen in the normal "horizontal" mode.</p> <p>Expanded mode is supported by all four output formats.</p> <p><code>\x</code> is the same as <code>\pset expanded</code>.</p>
fieldsep	<p>Specifies the field separator to be used in unaligned output mode. That way one can create, for example, tab- or comma-separated output, which other programs might prefer. To set a tab as field separator, type <code>\pset fieldsep '\t'</code>. The default field separator is <code>' '</code> (a vertical bar).</p>
footer	<p>Toggles the display of the default footer (<code>x rows</code>).</p>
null	<p>The second argument is a string that is printed whenever a column</p>

	is null. The default is not to print anything, which can easily be mistaken for, say, an empty string. Thus, one might choose to write <code>\pset null '(null)'</code> .
<code>recordsep</code>	Specifies the record (line) separator to use in unaligned output mode. The default is a newline character.
<code>trailingrecordsep</code>	Toggles on or off the trailing record separator to use in unaligned output mode.
<code>tuples_only</code> (or <code>t</code>)	Toggles between tuples only and full display. Full display might show extra information such as column headers, titles, and various footers. In tuples only mode, only actual table data is shown.
<code>title [text]</code>	Sets the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no argument is given, the title is unset.
<code>tableattr</code> (or <code>T</code>)[<code>text</code>]	Allows you to specify any attributes to be placed inside the HTML <code>table</code> tag. This could for example be <code>cellpadding</code> or <code>bgcolor</code> . Note that you probably don't want to specify <code>border</code> here, as that is already taken care of by <code>\pset border</code> .
<code>pager</code>	<p>Controls use of a pager for query and <code>vsq</code> help output. If the environment variable <code>PAGER</code> is set, the output is piped to the specified program. Otherwise a platform-dependent default (such as <code>more</code>) is used.</p> <p>When the pager is off, the pager is not used. When the pager is on, the pager is used only when appropriate; that is, the output is to a terminal and does not fit on the screen. (<code>vsq</code> does not do a perfect job of estimating when to use the pager.) <code>\pset pager</code> turns the pager on and off. Pager can also be set to <code>always</code>, which causes the pager to be always used.</p>

See illustrations on how these different formats look in the [Examples](#) section.

Tip: There are various shortcut commands for `\pset`. See `\a`, `\C`, `\H`, `\t`, `\T`, and `\x`.

`\q`

`\q` quits the `vsq` program.

`\qecho [STRING]`

`\qecho [STRING]` is identical to `\echo` except that the output is written to the query output stream, as set by `\o`.

`\r`

`\r` resets (clears) the query buffer.

For example, run the `\p` meta-command to see what is in the query buffer:

```
=> \p
CREATE VIEW my_seqview AS (SELECT * FROM sequences);
```

Now reset the query buffer:

```
=> \r
Query buffer reset (cleared).
```

If you reissue the command to see what's in the query buffer, you can see it is now empty:

```
=> \p
Query buffer is empty.
```

`\s [FILE]`

`\s [FILE]` prints or saves the command line history to *filename*. If a filename is not specified, `\s` writes the history to the standard output. This option is only available if `vsq` is configured to use the GNU Readline library.

`\set [NAME [VALUE [...]]]`

`\set [name [value [...]]]` sets the internal variable *name* to *value* or, if more than one value is given, to the concatenation of all of values. If no second argument is given, the variable is set with no value.

If no argument is provided, `\set` lists all internal variables; for example:

```
=> \set
VERSION = 'Vertica Analytic Database v6.0.0-0'
AUTOCOMMIT = 'off'
VERBOSITY = 'default'
PROMPT1 = '%/R%# '
PROMPT2 = '%/R%# '
PROMPT3 = '>> '
ROWS_AT_A_TIME = '1000'
DBNAME = 'VMartDB'
USER = 'dbadmin'
HOST = '<host_ip_address>'
PORT = '5433'
LOCALE = 'en_US@collation=binary'
HISTSIZE = '500'
```

Notes

- Valid variable names are case sensitive and can contain characters, digits, and underscores. vsql treats several variables as special, which are described in [Variables](#).
- The `\set` parameter `ROWS_AT_A_TIME` defaults to 1000. It retrieves results as blocks of rows of that size. The column formatting for the first block is used for all blocks, so in later blocks some entries could overflow. See [\timing](#) for examples.
- When formatting results, Vertica buffers `ROWS_AT_A_TIME` rows in memory to calculate the maximum column widths. It is possible that rows after this initial fetch are not properly aligned if any of the field values are longer than those seen in the first `ROWS_AT_A_TIME` rows. `ROWS_AT_A_TIME` can be [\unset](#) to guarantee perfect alignment, but this requires re-buffering the entire result set in memory and may cause vsql to fail if the result set is too big.
- To unset a variable, use the [\unset](#) command.

Using Backquotes to Read System Variables

In vsql, the contents of backquotes are passed to the system shell to be interpreted (the same behavior as many UNIX shells). This is particularly useful in setting internal vsql variables, since you may want to access UNIX system variables (such as `HOME` or `TMPDIR`) rather than hard-code values.

For example, if you want to set an internal variable to the full path for a file in your UNIX user directory, you could use backquotes to get the content of the system `HOME` variable, which is the full path to your user directory:

```
=> \set inputfile `echo $HOME`/myinput.txt=> \echo :inputfile  
/home/dbadmin/myinput.txt
```

The contents of the backquotes are replaced with the results of running the contents in a system shell interpreter. In this case, the `echo $HOME` command returns the contents of the `HOME` system variable.

\t

`\t` toggles the display of output column name headings and row count footer. This command is equivalent to `\pset tuples_only` and is provided for convenience.

\T [STRING]

`\T [STRING]` specifies attributes to be placed within the `table` tag in HTML tabular output mode. This command is equivalent to `\pset tableattr table_options`.

\timing

This meta-command reports, in milliseconds, the length of time each SQL statement runs. The results include:

- the length of time required to fetch the first block of rows
- the total time until the last block is formatted

For example:

```
=> SELECT * from allsales;  
state | name | sales  
-----  
NY    | B    | 20  
NY    | C    | 15  
MA    | A    | 40  
(3 rows)  
  
Time: First fetch (7 rows): 59.264 ms. All rows formatted: 59.362 ms
```

Timing is off by default. To determine the state of the timing function, enter the following:

```
=> \timing  
Timing is on
```

This example indicates that timing is off.

Toggle \timing function

You can set the timing state by entering on or off:

```
=> \timing on  
Timing is on
```

```
=> \timing off  
Timing is off
```

Alternatively, enter the function name without an argument to automatically toggle the state, whether it's on or off. For example, if you run `\timing on`, running `\timing` disables timing:

```
=> \timing on  
Timing is on  
  
=> \timing  
Timing is off
```

Enable \timing from the vsql Command Line

You can enable `\timing` from the command line using the `vsql -i` command. You can only use `-i` with the `-c` (command) and `-f` (filename) commands. For more information see [Command-Line Options](#).

From the command line enter the `-i` option before running a session to turn timing on. For example:

```
$VSQL -h host1 -U user1 -d VMart -p 15 -w ***** -i -f transactions.sql
```

```
$VSQL-h host1 -U user1 -d VMart -p 15 -w ***** -i -c "SELECT user_name,  
ssl_state, authentication_method, client_authentication_name, client_type FROM sessions  
WHERE session_id=(SELECT session_id FROM current_session);"
```

Example

The following example shows a SQL command with timing on:

```
=> \SELECT user_name, ssl_state, authentication_method, client_authentication_name,  
client_type FROM sessions WHERE session_id=(SELECT session_id FROM current_session);  
user_name | ssl_state | authentication_method | client_authentication_name | client_type
```

```
-----  
dbadmin | None | ImpTrust | default; Implicit Trust | vsql  
(1 row)
```

```
Time: First fetch (1 row): 44.388 ms. All rows formatted: 40.502 ms
```

The following table provides a quick guide on what occurs when you run `\timing` commands starting with the `\timing` being on:

vsql command	Starting <code>\timing</code> state = on
<code>=> \timing on</code>	Keeps timing on
<code>=> \timing</code>	Turns timing off
<code>=> \timing off</code>	Turns timing off
<code>=> \timing on</code>	Turns timing on

`\unset [NAME]`

`\unset [NAME]` unsets (deletes) the internal variable *name* that was set using the `\set` meta-command.

`\w [FILE]`

`\w [FILE]` outputs the current query buffer to the file *filename*.

`\x`

`\x` toggles extended table formatting mode. Is equivalent to `\pset expanded`.

Note: There is no space between the backslash and the x.

`\z`

`\z` lists table access privileges (grantee, grantor, privilege, and name) for all table access privileges in each schema. Is the same as `\dp`.

Variables

vsqL provides variable substitution features similar to common Linux command shells. Variables are simply name/value pairs, where the value can be any string of any length. To set variables, use the vsqL meta-command `\set` :

```
=> \set fact dim
```

sets the variable `fact` to the value `dim`. To retrieve the content of the variable, precede the name with a colon and use it as the argument of any slash command:

```
=> \echo :fact dim
```

Note: The arguments of `\set` are subject to the same substitution rules as with other commands. For example, `\set dim :fact` is a valid way to copy a variable.

If you call `\set` without a second argument, the variable is set, with an empty string as value. To unset (or delete) a variable, use the command `\unset` .

vsqL's internal variable names can consist of letters, numbers, and underscores in any order and any number. Some of these variables are treated specially by vsqL. They indicate certain option settings that can be changed at run time by altering the value of the variable or represent some state of the application. Although you can use these variables for any other purpose, this is not recommended. By convention, all specially treated variables consist of all upper-case letters (and possibly numbers and underscores). To ensure maximum compatibility in the future, avoid using such variable names for your own purposes.

SQL Interpolation

An additional useful feature of vsqL variables is that you can substitute ("interpolate") them into regular SQL statements. The syntax for this is again to prepend the variable name with a colon (:).

```
=> \set fact 'my_table'  
=> SELECT * FROM :fact;
```

would then query the table `my_table`. The value of the variable is copied literally (except for backquoted strings, see below), so it can even contain unbalanced quotes or backslash commands. Make sure that it makes sense where you put it. Variable interpolation is not performed into quoted SQL entities.

Note: The one exception to variable values being copied literally is strings in backquotes (`). The contents of backquoted strings are passed to a system shell, and replaced with the shell's output. See the [set](#) metacommmand topic for details.

AUTOCOMMIT

When AUTOCOMMIT is set 'on', each SQL command is automatically committed upon successful completion; for example:

```
\ set AUTOCOMMIT on
```

To postpone COMMIT in this mode, set the value as off.

```
\set AUTOCOMMIT off
```

If AUTOCOMMIT is empty or defined as off, SQL commands are not committed unless you explicitly issue COMMIT.

Notes

- AUTOCOMMIT is off by default.
- AUTOCOMMIT must be in uppercase, but the values, on or off, are case insensitive.
- In autocommit-off mode, you must explicitly abandon any failed transaction by entering ABORT or ROLLBACK.
- If you exit the session without committing, your work is rolled back.
- Validation on vsql variables is done when they are run, not when they are set.
- The [COPY](#) statement, by default, commits on completion, so it does not matter which AUTOCOMMIT mode you use, unless you issue COPY NO COMMIT. Please note that DDL statements are autocommitted.
- To tell if AUTOCOMMIT is on or off, issue the set command:

```
$ \set...  
AUTOCOMMIT = 'off'  
...
```

- AUTOCOMMIT is off if a `SELECT * FROM LOCKS` shows locks from the statement you just ran.

```
$ \set AUTOCOMMIT off
$ \set
...
AUTOCOMMIT = 'off'
...
SELECT COUNT(*) FROM customer_dimension;
 count
-----
 50000
(1 row)
SELECT node_names, object_name, lock_mode, lock_scope
FROM LOCKS;
 node_names |      object_name      | lock_mode | lock_scope
-----+-----+-----+-----
 site01     | Table:customer_dimension | S         | TRANSACTION
(1 row)
```

DBNAME

The name of the database to which you are currently connected. DBNAME is set every time you connect to a database (including program startup), but it can be unset.

ECHO

If set to `all`, all lines entered from the keyboard or from a script are written to the standard output before they are parsed or run.

To select this behavior on program start-up, use the switch `-a`. If set to `queries`, `vsq` merely prints all queries as they are sent to the server. The switch for this is `-e`.

ECHO_HIDDEN

When this variable is set and a backslash command queries the database, the query is first shown. This way you can study the Vertica internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch `-E`.)

If you set the variable to the value `noexec`, the queries are just shown but are not actually sent to the server and run.

ENCODING

The current client character set encoding.

HISTCONTROL

If this variable is set to `ignorespace`, lines that begin with a space are not entered into the history list. If set to a value of `ignoredups`, lines matching the previous history line are not entered. A value of `ignoreboth` combines the two options. If unset, or if set to any other value than those previously mentioned, all lines read in interactive mode are saved on the history list.

Source: Bash.

HISTSIZE

The number of commands to store in the command history. The default value is 500.

Source: Bash.

HOST

The database server host you are currently connected to. This is set every time you connect to a database (including program startup), but can be unset.

IGNOREEOF

If unset, sending an EOF character (usually Control+D) to an interactive session of `vsq` terminates the application. If set to a numeric value, that many EOF characters are ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

Source: Bash.

ON_ERROR_STOP

By default, if a script command results in an error, for example, because of a malformed command or invalid data format, processing continues. If you set `ON_ERROR_STOP` to 'on' in a script and an error occurs during processing, the script terminates immediately.

If you set `ON_ERROR_STOP` to 'on' in a script, run the script from Linux using `vsq1 -f <filename>`, and an error occurs, `vsq1` returns an error code 3 to Linux to indicate that the error occurred in a script.

To enable `ON_ERROR_STOP`:

```
=> \set ON_ERROR_STOP on
```

To disable `ON_ERROR_STOP`:

```
=> \set ON_ERROR_STOP off
```

PORT

The database server port to which you are currently connected. This is set every time you connect to a database (including program start-up), but can be unset.

PROMPT1 PROMPT2 PROMPT3

These specify what the prompts `vsq1` issues look like. See [Prompting](#) for details.

QUIET

This variable is equivalent to the command line option `-q`. It is probably not too useful in interactive mode.

SINGLELINE

This variable is equivalent to the command line option `-S`.

SINGLESTEP

This variable is equivalent to the command line option `-s`.

USER

The database user you are currently connected as. This is set every time you connect to a database (including program startup), but can be unset.

VERBOSITY

This variable can be set to the values `default`, `verbose`, or `tense` to control the verbosity of error reports.

VSQL_HOME

By default, the `vsql` program reads configuration files from the user's home directory. In cases where this is not desirable, the configuration file location can be overridden by setting the `VSQL_HOME` environment variable in a way that does not require modifying a shared resource.

In the following example, `vsql` reads configuration information out of `/tmp/jsmith` rather than out of `~`.

```
# Make an alternate configuration file in /tmp/jsmith
mkdir -p /tmp/jsmith
echo "\\echo Using VSQ_LRC in tmp/jsmith" > /tmp/jsmith/.vsq_lrc
# Note that nothing is echoed when invoked normally
vsq_l
# Note that the .vsq_lrc is read and the following is
# displayed before the vsq_l prompt
#
# Using VSQ_LRC in tmp/jsmith
VSQ_L_HOME=/tmp/jsmith vsq_l
```

VSQ_L_SSLMODE

`VSQ_L_SSLMODE` specifies how (or whether) clients (like `admintools`) use SSL when connecting to servers. The default value is `prefer`, meaning to use SSL if the server offers it. Legal values

are `require`, `prefer`, `allow`, and `disable`. This variable is equivalent to the command-line `-m` option (or `--sslmode`).

Prompting

The prompts `vsq` issues can be customized to your preference. The three variables `PROMPT1`, `PROMPT2`, and `PROMPT3` contain strings and special escape sequences that describe the appearance of the prompt. Prompt 1 is the normal prompt that is issued when `vsq` requests a new command. Prompt 2 is issued when more input is expected during command input because the command was not terminated with a semicolon or a quote was not closed. Prompt 3 is issued when you run a `SQL COPY` command and you are expected to type in the row values on the terminal.

The value of the selected prompt variable is printed literally, except where a percent sign (%) is encountered. Depending on the next character, certain other text is substituted instead. Defined substitutions are:

<code>%M</code>	The full host name (with domain name) of the database server, or <code>[local]</code> if the connection is over a socket, or <code>[local:/dir/name]</code> , if the socket is not at the compiled in default location.
<code>%m</code>	The host name of the database server, truncated at the first dot, or <code>[local]</code> .
<code>%></code>	The port number at which the database server is listening.
<code>%n</code>	The database session user name.
<code>%/</code>	The name of the current database.
<code>%~</code>	Like <code>%/</code> , but the output is <code>~</code> (tilde) if the database is your default database.
<code>%#</code>	If the session user is a database superuser, then a <code>#</code> , otherwise a <code>></code> . (The expansion of this value might change during a database session as the result of the command <code>SET SESSION AUTHORIZATION</code> .)
<code>%R</code>	In prompt 1 normally <code>=</code> , but <code>^</code> if in single-line mode, and <code>!</code> if the session is disconnected from the database (which can happen if <code>\connect</code> fails). In prompt 2 the sequence is replaced by <code>-</code> , <code>*</code> , a single quote, a double quote, or a dollar sign, depending on whether <code>vsq</code> expects more input because the command wasn't terminated yet, because you are inside a <code>/* ... */</code> comment, or because you are inside a quoted or dollar-escaped string. In prompt 3 the sequence doesn't produce anything.

<code>%x</code>	Transaction status: an empty string when not in a transaction block, or * when in a transaction block, or ! when in a failed transaction block, or ? when the transaction state is indeterminate (for example, because there is no connection).
<code>%digits</code>	The character with the indicated numeric code is substituted. If digits starts with 0x the rest of the characters are interpreted as hexadecimal; otherwise if the first digit is 0 the digits are interpreted as octal; otherwise the digits are read as a decimal number.
<code>%:name:</code>	The value of the vsql variable name. See the section Variables for details.
<code>%`command`</code>	The output of command, similar to ordinary "back- tick" substitution.
<code>%[... %]</code>	<p>Prompts may contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. In order for the line editing features of Readline to work properly, these non-printing control characters must be designated as invisible by surrounding them with %[and %]. Multiple pairs of these may occur within the prompt. The following example results in a boldfaced (1;) yellow-on-black (33;40) prompt on VT100-compatible, color-capable terminals:</p> <pre>testdb=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/R%#%[%033[0m%] '</pre> <p>To insert a percent sign into your prompt, write %%. The default prompts are '%/R%#' for prompts 1 and 2, and '>>' for prompt 3.</p> <p>Note: See the specification for terminal control sequences (applicable to gnome-terminal and xterm).</p>

Command Line Editing

vsql supports the tecla library for convenient line editing and retrieval.

The command history is automatically saved when vsql exits and is reloaded when vsql starts up. Tab-completion is also supported, although the completion logic makes no claim to be a SQL parser. If for some reason you do not like the tab completion, you can turn it off by putting this in a file named `.teclarc` in your home directory:

```
bind ^I
```

Read the tecla documentation for further details.

Notes

The vsql implementation of the tecla library deviates from the tecla documentation as follows:

- Recalling Previously Typed Lines

Under pure tecla, all new lines are appended to a list of historical input lines maintained within the GetLine resource object. In vsql, only different, non-empty lines are appended to the list of historical input lines.

- History Files

tecla has no standard name for the history file. In vsql, the file name is called `~/.vsql_hist`.

- International Character Sets (Meta keys and locales)

In vsql, 8-bit meta characters are no longer supported. Make sure that meta characters send an escape by setting their EightBitInput X resource to False. You can do this in one of the following ways:

- Edit the `~/.Xdefaults` file by adding the following line:

```
XTerm*EightBitInput: False
```

- Start an xterm with an `-xrm '*EightBitInput: False'` command-line argument.

- Key Bindings:

- The following key bindings are specific to vsql:

- *Insert* switches between insert mode (the default) and overwrite mode.
- *Delete* deletes the character to the right of the cursor.
- *Home* moves the cursor to the front of the line.
- *End* moves the cursor to the end of the line.
- `^R` Performs a history backwards search.

vsq Environment Variables

Set one or more of the following environment variables to be used by the defined properties automatically, each time you start vsq:

- **PAGER** - If the query results do not fit on the screen, they are piped through this command. Typical values are `more` or `less`. The default is platform-dependent. The use of the pager can be disabled by using the `\pset` command.
- **VSQ_HOME** - By default, the vsq program reads configuration files from the user's home directory. In cases where this is not desirable, the configuration file location can be overridden by setting the VSQ_HOME environment variable in a way that does not require modifying a shared resource. For more information, see [VSQ_HOME](#).
- **VSQ_HOST** - Host name or IP address of the Vertica node.
- **VSQ_PORT** - Port to use for the connection.
- **VSQ_USER** - Username to use for the connection.
- **VSQ_PASSWORD** - The database password. Using this environment variable increases site security by precluding the need to enter the database password on the command line.
- **VSQ_EDITOR**, **EDITOR** and **VISUAL** - Editor used by the `\e` command. The variables are examined in the order listed; the first that is set is used.
- **VSQ_SSLMODE** - Specifies how (or whether) clients (like admintools) use SSL when connecting to servers. For more information, see [VSQ_SSLMODE](#).
- **TMPDIR** - Directory for storing temporary files. The default is platform-dependant. On Unix-like systems the default is `/tmp`.

Locales

The default terminal emulator under Linux is `gnome-terminal`, although `xterm` can also be used.

Micro Focus recommends that you use `gnome-terminal` with vsq in UTF-8 mode, which is its default.

To Change Settings on Linux

1. From the tabs at the top of the vsql screen, select Terminal.
2. Click **Set Character Encoding**.
3. Select **Unicode (UTF-8)**.

Note: This works well for standard keyboards. xterm has a similar UTF-8 option.

To Change Settings on Windows Using PuTTY

1. Right click the vsql screen title bar and select **Change Settings**.
2. Click **Window** and click **Translation**.
3. Select **UTF-8** in the drop-down menu on the right.

Notes

- vsql has no way of knowing how you have set your terminal emulator options.
- The tecla library is prepared to do POSIX-type translations from a local encoding to UTF-8 on interactive input, using the POSIX LANG, etc., environment variables. This could be useful to international users who have a non-UTF-8 keyboard. See the tecla documentation for details.

Micro Focus recommends the following (or whatever other .UTF-8 locale setting you find appropriate):

```
export LANG=en_US.UTF-8
```

- The vsql `\locale` command invokes and tracks the server `SET LOCALE TO` command, described in the SQL Reference Manual. vsql itself currently does nothing with this locale setting, but rather treats its input (from files or from tecla), all its output, and all its interactions with the server as UTF-8. vsql ignores the POSIX locale variables, except for any "automatic" uses in `printf`, and so on.

Files

Before starting up, `vsq` attempts to read and execute commands from the system-wide `vsq_lrc` file and the user's `~/vsq_lrc` file. The command-line history is stored in the file `~/vsq_history`.

Tip: If you want to save your old history file, open another terminal window and save a copy to a different file name.

Exporting Data Using `vsq`

You can use `vsq` for simple data-export tasks by changing its output format options so the output is suitable for importing into other systems (tab-delimited or comma-separated files, for example). These options can be set either from within an interactive `vsq` session, or through command-line arguments to the `vsq` command (making the export process suitable for automation through scripting). After you have set `vsq`'s options so it outputs the data in a format your target system can read, you run a query and capture the result in a text file.

The following table lists the meta-commands and command-line options that are useful for changing the format of `vsq`'s output.

Description	Meta-command	Command-line Option
Disable padding used to align output.	<code>\a</code>	<code>-A</code> or <code>--no-align</code>
Show only tuples, disabling column headings and row counts.	<code>\t</code>	<code>-t</code> or <code>--tuples-only</code>
Set the field separator character.	<code>\pset fieldsep</code>	<code>-F</code> or <code>--field-separator</code>
Send output to a file.	<code>\o</code>	<code>-o</code> or <code>--output</code>
Specify a SQL statement to execute.	N/A	<code>-c</code> or <code>--command</code>

The following example demonstrates disabling padding and column headers in the output, and setting a field separator to dump a table to a tab-separated text file within an interactive session.

```
=> SELECT * FROM my_table;
 a |   b   | c
---+-----+---
 a | one   | 1
 b | two   | 2
 c | three | 3
 d | four  | 4
 e | five  | 5
(5 rows)
=> \a
Output format is unaligned.
=> \t
Showing only tuples.
=> \pset fieldsep '\t'
Field separator is "    ".
=> \o dumpfile.txt
=> select * from my_table;
=> \o
=> \! cat dumpfile.txt
a      one      1
b      two      2
c      three    3
d      four     4
e      five     5
```

Note: You could encounter issues with empty strings being converted to NULLs or the reverse using this technique. You can prevent any confusion by explicitly setting null values to output a unique string such as NULLNULLNULL (for example, `\pset null 'NULLNULLNULL'`). Then, on the import end, convert the unique string back to a null value. For example, if you are copying the file back into a Vertica database, you would give the argument `NULL 'NULLNULLNULL'` to the [COPY](#) statement.

When logged into one of the database nodes, you can create the same output file directly from the command line by passing the right parameters to `vsql`:

```
$ vsql -U username -F $'\t' -At -o dumpfile.txt -c "SELECT * FROM my_table;"
Password:
$ cat dumpfile.txt
a      one      1
b      two      2
c      three    3
d      four     4
e      five     5
```

If you want to convert null values to a unique string as mentioned earlier, you can add the argument `-P null='NULLNULLNULL'` (or whatever unique string you choose).

By adding the `-w vsql` command-line option to the example command line, you could use the command within a batch script to automate the data export. However, the script would contain the database password as plain text. If you take this approach, you should prevent unauthorized access to the batch script, and also have the script use a database user account that has limited access.

To set the field separator value to a control character, use your shell's control character escape notation. In Bash, you specify a control character in an argument using a dollar sign (\$) followed by a string contained in single quotes. This string can contain C-string escapes (such as \t for tab), or a backslash (\) followed by an octal value for the character you want to use.

The following example demonstrates setting the separator character to tab (\t), vertical tab (\v) and the octal value of vertical tab (\013).

```
$ vsql -At -c "SELECT * FROM testtable;"
A|1|2|3
B|4|5|6

$ vsql -F $'\t' -At -c "SELECT * FROM testtable;"
A      1      2      3
B      4      5      6

$ vsql -F $'\v' -At -c "SELECT * FROM testtable;"
A
  1
  2
  3
B
  4
  5
  6

$ vsql -F $'\013' -At -c "SELECT * FROM testtable;"
A
  1
  2
  3
B
  4
  5
  6
```

Copying Data Using vsql

You can use vsql to copy data between two Vertica databases. This technique is similar to the technique explained in [Exporting Data Using vsql](#), except instead of having vsql save data to a file for export, you pipe one vsql's output to the input of another vsql command that runs a [COPY](#) statement from STDIN. This technique can also work for other databases or applications that accept data from an input stream.

Note: The following technique only works for individual tables. To copy an entire database to another cluster, see [Copying the Database to Another Cluster](#) in the Administrator's Guide.

The easiest way to copy using `vsql` is to log in to a node of the target database, then issue a `vsql` command that connects to the source Vertica database to dump the data you want. For example, the following command copies the `store.store_sales_fact` table from the `vmart` database on node `testdb01` to the `vmart` database on the node you are logged into:

```
vsql -U username -w passwd -h testdb01 -d vmart -At -c "SELECT * from store.store_sales_fact" \
| vsql -U username -w passwd -d vmart -c "COPY store.store_sales_fact FROM STDIN DELIMITER '|';"
```

Note: The above example copies the data only, not the table design. The target table for the data copy must already exist in the target database. You can export the design of the table using [EXPORT_OBJECTS](#) or [EXPORT_CATALOG](#).

If you are using the Bash shell, you can escape special delimiter characters. For example, `DELIMITER E'\t'` specifies tab. Shells other than Bash may have other string-literal syntax.

Monitoring Progress (optional)

You may want some way of monitoring progress when copying large amounts of data between Vertica databases. One way of monitoring the progress of the copy operation is to use a utility such as [Pipe Viewer](#) that pipes its input directly to its output while displaying the amount and speed of data it passes along. Pipe Viewer can even display a progress bar if you give it the total number of bytes or lines you expect to be processed. You can get the number of lines to be processed by running a separate `vsql` command that executes a [SELECT COUNT](#) query.

Note: Pipe Viewer isn't a standard Linux or Solaris command, so you will need to download and install it yourself. See the [Pipe Viewer](#) page for download packages and instructions. Micro Focus does not support Pipe Viewer. Install and use it at your own risk.

The following command demonstrates how you can use Pipe Viewer to monitor the progress of the copy shown in the prior example. The command is complicated by the need to get the number of rows that will be copied, which is done using a separate `vsql` command within a Bash backquote string, which executes the string's contents and inserts the output of the command into the command line. This `vsql` command just counts the number of rows in the `store.store_sales_fact` table.

```
vsql -U username -w passwd -h testdb01 -d vmart -At -c "SELECT * from store.store_sales_fact" \
| pv -lpetr -s `vsql -U username -w passwd -h testdb01 -d vmart -At -c "SELECT COUNT (*) FROM
store.store_sales_fact;"` \
| vsql -U username -w passwd -d vmart -c "COPY store.store_sales_fact FROM STDIN DELIMITER '|';"
```

While running, the above command displays a progress bar that looks like this:

```
0:00:39 [12.6M/s] [=====>] ] 50% ETA 00:00:40
```

Output Formatting Examples

The first example shows how to spread a command over several lines of input. Notice the changing prompt:

```
=> CREATE TABLE my_table (  
-> first integer not null default 0,  
-> second char(10));  
CREATE TABLE
```

Assume you have filled the table with data and want to take a look at it:

```
testdb=> SELECT * FROM my_table;  
first | second  
-----+-----  
1 | one  
2 | two  
3 | three  
4 | four  
(4 rows)
```

You can display tables in different ways by using the `\pset` command:

```
testdb=> \pset border 2  
Border style is 2.  
testdb=> SELECT * FROM my_table;  
+-----+-----+  
| first | second |  
+-----+-----+  
| 1 | one |  
| 2 | two |  
| 3 | three |  
| 4 | four |  
+-----+-----+  
(4 rows)  
  
=> \pset border 0  
Border style is 0.  
=> SELECT * FROM my_table;  
first second  
-----  
1 one  
2 two  
3 three  
4 four  
(4 rows)  
  
=> \pset border 1 Border style is 1.  
=> \pset format unaligned  
Output format is unaligned.  
=> \pset fieldsep ','  
Field separator is ",".  
=> \pset tuples_only
```

```
Showing only tuples.  
=> SELECT second, first FROM my_table;  
one,1  
two,2  
three,3  
four,4
```

Alternatively, use the short commands:

```
=> \a \t \x Output format is aligned.  
Tuples only is off.  
Expanded display is on.  
=> SELECT * FROM my_table;  
first | 1  
second | one  
-----+-----  
first | 2  
second | two  
-----+-----  
first | 3  
second | three  
-----+-----  
first | 4  
second | four
```

Client Libraries

The Vertica client driver libraries provide interfaces for connecting your client applications (or third-party applications such as Cognos and MicroStrategy) to your Vertica database. The drivers simplify exchanging data for loading, report generation, and other common database tasks.

There are three separate client drivers:

- Open Database Connectivity (ODBC)—the most commonly-used interface for third-party applications and clients written in C, Python, PHP, Perl, and most other languages.
- Java Database Connectivity (JDBC)—used by clients written in the Java programming language.
- ActiveX Data Objects for .NET (ADO.NET)—used by clients developed using Microsoft's .NET Framework and written in C#, Visual Basic .NET, and other .NET languages.

Client Driver Standards

The Vertica client drivers are compatible with the following driver standards:

- The ODBC driver complies with version 3.5.1 of the ODBC standard.
- Vertica's JDBC driver is a type 4 driver that complies with the JDBC 3.0 standard. It is compiled using JDK version 1.5, and is compatible with client applications compiled using JDK versions 1.5 and 1.6.
- ADO.NET drivers conform to .NET framework 3.0 specifications.

The drivers do not support some of the optional features in the standards. See [ODBC Feature Support](#) and [JDBC Feature Support](#) and [Using ADO.NET](#) for details.

Client Driver and Server Version Compatibility

Usually, each version of the Vertica server is compatible with the previous version of the client drivers. This compatibility lets you upgrade your Vertica server without having to immediately upgrade your client software. However, some new features of the new server version may not be available through the old drivers.

Note: Vertica Release 8.1.x and later adds backwards compatibility for the ODBC client driver. The 8.1.x ODBC client driver is backwards compatible to Vertica server version 7.1. For full compatibility with the previous server version, specify the Protocol property in your connection string. For more information about the Protocol property, see [Data Source Name \(DSN\) Connection Properties](#).

The following tables summarize the compatibility of each recent version of the client drivers with the Vertica server versions.

The following table indicates that, in general, all clients are forward compatible.

Client	Client Driver Version	Compatible Server Versions
All Clients	6.1.x	6.1.x, 7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x
	7.0.x	7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x
	7.1.x	7.1.x, 7.2.x, 8.0.x, 8.1.x
	7.2.x	7.2.x, 8.0.x, 8.1.x
	8.0.x	8.0.x, 8.1.x
	8.1.x	8.1.x

The following table lists FIPS 140-2 compatible clients.

Client	Client Driver Version	Compatible Server Versions
FIPS-enabled ODBC	8.0.x	8.0.x, 8.1.x
FIPS-enabled ODBC	8.1.x	8.0.x, 8.1.x
FIPS-enabled JDBC	8.1.x	8.1.x

The following table indicates that the 8.1.x ODBC client is backward compatible.

Client	Client Driver Version	Compatible Server Versions
ODBC (backwards compatibility)	8.1.x	7.1.x, 7.2.x, 8.0.x, 8.1.x

Vertica ODBC/JDBC Client Installers

The ODBC/JDBC client drivers are a separate installation from the ADO.NET drivers. (ADO.NET support is not available in Community Edition.) As noted in the compatibility table, the 6.x ODBC/JDBC client drivers do not support access to a non Vertica 6.x database and above. For example, you cannot use the new 6.x ODBC/JDBC client drivers to access a Vertica 5.x database. If you plan on having a mixed Vertica environment supporting both 5.x and 6.x Vertica database, consider keeping the 5.x drivers installed.

ODBC/JDBC Multiple Version Installations

The following ODBC/JDBC drivers are supported on a single machine:

- 4.x and 5.x ODBC/JDBC drivers can be installed on the same machine.
- 4.x and 6.x ODBC/JDBC drivers can be installed on the same machine.

It is not possible to have both 5.x and 6.x ODBC drivers on a single machine. If you install the 6.x version, it automatically overlays the existing 5.x installation, and any DSN defined against a 5.x Vertica database is not supported.

Vertica ADO.NET Client Installers

Prior to version 6.x, ADO.Net drivers must be uninstalled prior to installing a later version of the driver. The 6.x ADO.Net drivers require the Vertica database to be 6.0.0 or above. The ADO.NET 6.x driver only supports access to a Vertica 6.x server. The ADO.NET 4.x plug-in does not work with a Vertica 6.x server. If you plan on also using the ODBC bridge and you need to access both Vertica 5.x and 6.x databases, consider keeping the 5.x versions of the ODBC/JDBC drivers for the reasons stated previously.

Client Drivers

You must install the Vertica client drivers to access Vertica from your client application. The drivers create and maintain connections to the database and provide APIs that your applications use to access your data. The client drivers support connections using JDBC, ODBC, and [ADO.NET](#).

Client Driver Standards

The client drivers support the following standards:

- ODBC drivers conform to ODBC 3.5.1 specifications.
- JDBC drivers conform to JDK 5 specifications.
- ADO.NET drivers conform to .NET framework 3.0 specifications.

Installing the Client Drivers

How you install client drivers depends on the client's operating system:

- For Linux and UNIX clients, you must first [install a Linux driver manager](#). After you have installed the driver manager, there are two different ways to install the client drivers:
 - On Red Hat Enterprise Linux 5, 64-bit and SUSE Linux Enterprise Server 10/11 64-bit, you can use the Vertica client RPM package to install the ODBC and JDBC drivers as well as the vsql client.
 - On other Linux platforms and UNIX-like platforms you can download the ODBC and JDBC drivers and install them individually.

Note: The ODBC and JDBC client drivers are installed by the server `.rpm` files. If you have installed Vertica Analytics Platform on your Linux system for development or testing purposes, you do not need to download and install the client drivers on it—you just need to configure the drivers. To use ODBC, you need to create a DSN (see [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)). To use JDBC, you need to add the JDBC

client driver to the Java CLASSPATH (see [Modifying the Java CLASSPATH](#)). (The JDBC client driver is not available on FIPS-compliant systems.)

- On Mac OS X clients, download the ODBC client driver .pkg file. The driver is compatible with both 32-bit and 64-bit applications.
- On Windows clients, download the 32-bit or 64-bit client installer. The installer provides the ODBC client driver, the ADO.NET client driver, the OLE DB client driver, the vsqll client, the Microsoft Connectivity Pack, and the Visual Studio plug-in.
- There is a cross-platform JDBC client driver .jar file available for installation on all platforms.

The remainder of this section explain the requirements for the Vertica client drivers, and the procedure for downloading, installing, and configuring them.

Driver Prerequisites

The following topics describe the system requirements for the client drivers. You need to ensure that your client system meets these requirements before installing and using the client drivers.

ODBC Prerequisites

There are several requirements your client systems must meet before you can install the Vertica ODBC drivers.

Operating System

The Vertica ODBC driver requires a supported platform. The list of currently-supported platforms can be found at [Vertica 8.1.x Client Drivers](#).

ODBC Driver Manager

The Vertica ODBC driver requires that the client system have a supported driver manager. See [Installing Driver Managers Linux and Other UNIX-like Platforms](#) for details.

UTF-8, UTF-16 and UTF-32 Support

The Vertica ODBC driver is a universal driver that supports UTF-8, UTF-16, and UTF-32 encoding. The default setting depends on the client platform. For details, see [Required ODBC Driver Configuration Settings for Linux and UNIX](#).

When using the driver with the DataDirect Connect driver manager, DataDirect Connect adapts to the ODBC driver's text encoding settings. You should configure the ODBC driver to use the encoding method that your application requires. This allows strings to be passed between the driver and the application without intermediate conversion.

See Also

- [Client Drivers](#)
- [Programming ODBC Client Applications](#)
- [Creating an ODBC Data Source Name \(DSN\)](#)

ADO.NET Prerequisites

The Vertica driver for ADO.NET requires the following software and hardware components:

Operating System

The Vertica ADO.NET driver requires a supported Windows operating system. The list of supported platforms can be found in the Supported Platforms document at <http://my.vertica.com/docs>.

Memory

Vertica suggests a minimum of 512MB of RAM.

.NET Framework

The requirements for the .NET framework for ADO.NET in Vertica can be found in the Supported Platforms document at <http://my.vertica.com/docs>.

See Also

- [Programming ADO.NET Applications](#)

Python Prerequisites

Python is a free, agile, object-oriented, cross-platform programming language designed to emphasize rapid development and code readability. Python has been released under several different open source licenses.

Vertica's ODBC driver is tested with multiple versions of Python. See [Perl and Python Requirements](#) for details.

Python Driver

Vertica requires the Vertica Python Client or the pyodbc driver module. See your system's Python documentation for installation and configuration information.

Supported Operating Systems

The Vertica ODBC driver requires one of the operating systems listed in [ODBC Prerequisites](#). For usage and examples, see [Programming Python Client Applications](#).

Perl Prerequisites

Perl is a free, stable, open source, cross-platform programming language licensed under its Artistic License, or the GNU General Public License (GPL).

Your Perl scripts access Vertica through its ODBC driver, using the Perl Database Interface (DBI) module with the ODBC Database Driver (DBD::ODBC). The Vertica ODBC driver is known to be compatible with these versions of Perl:

- 5.8
- 5.10

Later Perl versions may also work.

Perl Drivers

The following Perl driver modules have been tested with the Vertica ODBC driver:

- The DBI driver module, version 1.609
- The DBD::ODBC driver module, version 1.22

Other versions may also work.

Supported Client Systems

The Vertica ODBC driver requires one of the operating systems and driver managers listed in [ODBC Prerequisites](#).

PHP Prerequisites

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. PHP is licensed under the PHP License, an open-source BSD-style license certified by the Open Source Initiative.

PHP Modules

The following PHP modules are required:

- php
- php-odbc
- php-pdo
- UnixODBC (if you are using the Unix ODBC driver)
- libiodbc (if you are using the iODBC driver)

Supported Client Systems

The Vertica ODBC driver requires one of the operating systems and driver managers listed in [ODBC Prerequisites](#).

Upgrading the Client Drivers

The Vertica client drivers are usually updated for each new release of the Vertica server. The client driver installation packages include the version number of the corresponding Vertica server release. Usually, the drivers are forward-compatible with the next release, so your client applications are still be able to connect using the older drivers after you upgrade to the next version of Vertica Analytics Platform server. See [Client Driver and Server Version Compatibility](#) for details on which client driver versions work with each version of Vertica server.

Note: Vertica Release 8.1.x and later adds backwards compatibility for the ODBC client driver. The 8.1.x ODBC client driver is backwards compatible to Vertica server version 7.1. For full compatibility with the previous server version, specify the Protocol property in your connection string. For more information about the Protocol property, see [Data Source Name \(DSN\) Connection Properties](#).

You should upgrade your clients as soon as possible after upgrading your server, to take advantage of new features and to maintain maximum compatibility with the server.

To upgrade your drivers, follow the same procedure you used to install them in the first place. The new installation will overwrite the old. See the specific instructions for installing the drivers on your client platform for any special instructions regarding upgrades.

Note: Installing new ODBC drivers does not alter existing DSN settings. You may need to change the driver settings in either the DSN or in the `odbcinst.ini` file, if your client system uses one. See [Creating an ODBC Data Source Name](#) for details.

Setting a Client Connection Label

You can set a client connection label when you connect to a Vertica database. You can also set or return the client connection label using the [SET_CLIENT_LABEL](#) and [GET_CLIENT_LABEL](#) functions.

Set the client connection label:

```
=> SELECT SET_CLIENT_LABEL('py_data_load_application');
      SET_CLIENT_LABEL
-----
client_label set to py_data_load_application
(1 row)
```

Return the current client connection label:

```
=> SELECT GET_CLIENT_LABEL();
       GET_CLIENT_LABEL
-----
py_data_load_application
(1 row)
```

JDBC

The JDBC Client has a method to set and return the client connection label: `getClientInfo()` and `setClientInfo()`. You can use these methods with the SQL Functions [GET_CLIENT_LABEL](#) and [SET_CLIENT_LABEL](#).

When you use these two methods, make sure you pass the string value `APPLICATIONNAME` to both the setter and getter methods.

Use `setClientInfo()` to create a client label, and use `getClientInfo()` to return the client label:

```
import java.sql.*;
import java.util.Properties;

public class ClientLabelJDBC {

    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "");
        myProp.put("loginTimeout", "35");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://docc05.verticacorp.com:5433/doccdb", myProp);
            System.out.println("Connected!");
            conn.setClientInfo("APPLICATIONNAME", "JDBC Client - Data Load");
            System.out.println("New Conn label: " + conn.getClientInfo("APPLICATIONNAME"));
            conn.close();
        } catch (SQLException connException) {
            // There was a potentially temporary network error
            // Could automatically retry a number of times here, but
            // instead just report error and exit.
            System.out.print("Network connection issue: ");
            System.out.print(connException.getMessage());
            System.out.println(" Try again later!");
            return;
        } catch (SQLException authException) {
            // Either the username or password was wrong
            System.out.print("Could not log into database: ");
            System.out.print(authException.getMessage());
            System.out.println(" Check the login credentials and try again.");
            return;
        } catch (SQLException e) {
            // Catch-all for other exceptions
            e.printStackTrace();
        }
    }
}
```

```
}
```

When you run this method, it prints the following result to the standard output:

```
Connected!  
New Conn Label: JDBC Client - Data Load
```

Additional Parameter Settings

The following parameters can be set for the Vertica client drivers.

Logging Settings

These parameters control how messages between the client and server are logged. None of these settings are required. If they are not set, then the client library does not log any messages. They apply to both ADO.NET and ODBC.

- **LogLevel**—The severity of messages that are logged between the client and the server. The valid values are:
 - 0—No logging
 - 1—Fatal errors
 - 2—Errors
 - 3—Warnings
 - 4—Info
 - 5—Debug
 - 6—Trace (all messages)

The value you specify for this setting sets the minimum severity for a message to be logged. For example, setting LogLevel to 3 means that the client driver logs all warnings, errors, and fatal errors.

Note: On a Windows client, you have the option of directing ODBC or OLE DB log entries to Event Tracing for Windows (ETW). Once set, ODBC log entries appear in the Windows Event Viewer. See [Register the ODBC Driver as a Windows Event Log Provider, and](#)

[Enable the Logs](#) for ODBC, or [Register the OLE DB Driver as a Windows Event Log Provider, and Enable the Logs](#) for OLE DB.

- **LogPath**—The absolute path of a directory to store log files . For example:
`/var/log/verticaodbc`
- **LogNamespace**—Limits logging to messages generated by certain objects in the client driver.

Note: These settings are also available for the Vertica JDBC driver through connection properties. See [Connection Properties](#) for details.

ODBC-specific Settings

The following settings are used only by the Vertica ODBC client driver.

- **DriverManagerEncoding**—The UTF encoding standard that the driver manager uses. This setting needs to match the encoding the driver manager expects. The available values for this setting are:
 - UTF-8
 - UTF-16 (usually used by unixODBC)
 - UTF-32 (usually used by iODBC)

See the documentation for your driver manager to find the correct value for this setting.

Note: While both UTF-16 and UTF-8 are valid settings for DataDirect, Vertica recommends that you set the DataDirect driver manager encoding to UTF-16.

If you do not set this parameter, the ODBC driver defaults to the value shown in the following table. If your driver manager uses a different encoding, you must set this value for the ODBC driver to be able to work.

Client Platform	Default Encoding
AIX 64-bit	UTF-32
HP-UX 64-bit	UTF-32

Client Platform	Default Encoding
Linux 32-bit	UTF-32
Linux 64-bit	UTF-32
Linux Itanium 64-bit	UTF-32
OS X	UTF-32
Solaris 64-bit	UTF-32
Solaris SPARC 64-bit	UTF-32
Windows 32-bit	UTF-16
Windows 64-bit	UTF-16

- **ErrorMessagesPath**—The absolute path to the parent directory that contains the Vertica client driver's localized error message files. These files are usually stored in the same directory as the Vertica ODBC driver files.

Note: This setting is required. If you do not set it, then any error the ODBC driver encounters will result in an error message about a missing `ODBCMessages.xml` file.

- **ODBCInstLib**—The absolute path to the file containing the ODBC installer library (`ODBCInst`). This setting is required if the directory containing this library is not set in the `LD_LIBRARY_PATH` or `LIB_PATH` environment variables. The library files for the major driver manager are:
 - UnixODBC: `libodbcinst.so`
 - iODBC: `libiodbcinst.so` (`libiodbcinst.2.dylib` on OS X)
 - DataDirect: `libodbcinst.so`

Note: On AIX platforms, you need give the path to the library archive, followed by the name of the library enclosed in parenthesis. For example:
`ODBCInstLib=/usr/lib64/libodbcinst.a(libodbcinst.so.1)`

ADO.NET-specific Settings

This setting applies only to the ADO.NET client driver:

C#PreloadLogging—Tells the Vertica ADO.NET driver to begin logging as soon as possible, before the driver has fully loaded itself. Normally, logging only starts after the driver has fully loaded. Valid values for this setting are:

- 0—Do not start logging before the driver has loaded.
- 1—Start logging as soon as possible.

Using Legacy Drivers

The Vertica server supports connections from the previous version of the client drivers. For example, the Vertica version 5.1 server works with the 4.1 client drivers, since they were the drivers distributed with the previous version of the server. This backwards compatibility lets you upgrade your Vertica database first, then later upgrade your clients.

If you have not yet updated your code to work with the new version of the Vertica client drivers, you can continue to use the older drivers until you do. If you need to install your client application on a new client system, you can download and install the older drivers. See [myVertica portal](#) to download the installers; find installation documentation at <http://my.vertica.com/docs>.

For detailed information on which the compatibility of different versions of the Vertica server and Vertica client, see [Client Driver and Server Version Compatibility](#).

Note: The support for a previous version of the drivers is usually eliminated in the next release of Vertica. For example, the Vertica version 5.1 server does not support the version 4.0 drivers. You should update your client application to work with the new client drivers as soon as possible.

Modifying the Java CLASSPATH

The CLASSPATH environment variable contains the list of directories where the Java run time looks for library class files. For your Java client code to access Vertica, you need to add the directory where the Vertica JDBC .jar file is located.

Note: In your CLASSPATH, use the symbolic link `vertica-jdbc-x.x.x.jar` (where `x.x.x` is a version number) that points to the JDBC library .jar file, rather than the .jar file itself. Using the symbolic link ensures that any updates to the JDBC library .jar file (which will use a different filename) will not invalidate your CLASSPATH setting, since the symbolic link's filename will remain the same. You just need to update the symbolic link to point at the new .jar file.

Linux, Solaris, AIX, HP-UX, and OS X

If you are using the Bash shell, use the `export` command to define the `CLASSPATH` variable:

```
# export CLASSPATH=/opt/vertica/java/lib/vertica-jdbc-x.x.x.jar
```

If environment variable `CLASSPATH` is already defined, use the following command to prevent it from being overwritten:

```
# export CLASSPATH=$CLASSPATH:/opt/vertica/java/lib/vertica-jdbc-x.x.x.jar
```

If you are using a shell other than Bash, consult its documentation to learn how to set environment variables.

You need to either set the `CLASSPATH` environment variable for every login session, or insert the command to set the variable into one of your startup scripts (such as `~/ .profile` or `/etc/profile`).

Windows

Provide the class paths to the `.jar`, `.zip`, or `.class` files.

```
C:> SET CLASSPATH=classpath1;classpath2...
```

For example:

```
C:> SET CLASSPATH=C:\java\MyClasses\vertica-jdbc-x.x.x.jar
```

As with the Linux/UNIX settings, this setting only lasts for the current session. To set the `CLASSPATH` permanently, set an environment variable:

1. On the Windows Control Panel, click **System**.
2. Click **Advanced** or **Advanced Systems Settings**.
3. Click **Environment Variables**.
4. Under User variables, click **New**.
5. In the Variable name box, type `CLASSPATH`.
6. In the Variable value box, type the path to the Vertica JDBC `.jar` file on your system (for example, `C:\Program Files (x86)\Vertica\JDBC\vertica-jdbc-x.x.x.jar`)

Specifying the Library Directory in the Java Command

There is an alternative way to tell the Java run time where to find the Vertica JDBC driver other than changing the CLASSPATH environment variable: explicitly add the directory containing the .jar file to the java command line using either the `-cp` or `-classpath` argument. For example, on Linux, start your client application using:

```
# java -classpath /opt/vertica/java/lib/vertica-jdbc-x.x.x.jar myapplication.class
```

Your Java IDE may also let you add directories to your CLASSPATH, or let you import the Vertica JDBC driver into your project. See your IDE documentation for details.

Installing the Client Drivers on Linux and UNIX-Like Platforms

This topic details how to install the client drivers on Linux and Unix-like platforms.

Installing Driver Managers Linux and Other UNIX-like Platforms

If your client platform does not already have an ODBC driver manager, you need to install one before you can use the Vertica ODBC client driver. The driver manager provides an interface between your client operating system and the ODBC drivers. See [Vertica 8.1.x Client Drivers](#) for a list of driver managers that are supported on each of the client platforms.

Driver managers can be downloaded from your operating system specific repository and from the links below.

Vertica does not provide instructions for installing and configuring these third party binaries. For download and configuration information, see the respective websites for the driver managers for installation and configuration information:

- UnixODBC: <http://www.unixodbc.org/>
- iODBC: <http://www.iodbc.org>

Installing the Client RPM on Red Hat and SUSE

For Red Hat Enterprise Linux and SUSE Linux Enterprise Server, you can download and install a client driver RPM that installs both the ODBC and JDBC driver libraries and the vsql client.

Important: Vertica provides the FIPS-compliant client driver only as an rpm for 64-bit clients. You can install this rpm only on FIPS-enabled machines. The FIPS client includes vsql and ODBC drivers. If you are installing the FIPS-specific client, refer to the section, [Installing the FIPS Client Driver for ODBC and vsql](#).

To install the client driver RPM package:

1. Open a Web browser and log in to the [myVertica portal](#).
2. Click Downloads, and choose Client Drivers.
3. Download the client RPM file that matches your client platform's architecture.

Note: The 64-bit client driver RPM installs both the 64-bit and 32-bit ODBC driver libraries on non-FIPS compliant systems.

4. If you did not directly download the RPM on the client system, transfer the file to the client.
5. Log in to the client system as root.
6. Install the RPM package you downloaded:

```
# rpm -Uvh package_name.rpm
```

Note: You receive one or more conflict error messages if there are existing Vertica client driver files on your system. This can happen if you are trying to install the client driver package on a system that has the server package installed, since the server package also includes the client drivers. In this case, you don't need to install the client drivers, and can instead use the drivers installed by the server package. If the conflict arises from an old driver installation or from a server installation for an older version, you can use the rpm command's `--force` switch to force it to overwrite the existing files with the files in the client driver package.

Once you have installed the client package, you need to create a DSN (see [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)) and set some additional configuration parameters (see [Required ODBC Driver Configuration Settings for Linux and UNIX](#)) to use ODBC. To use JDBC, you need to modify your class path (see [Modifying the Java CLASSPATH](#)) before you can use JDBC.

You may also want to add the vsql client to your PATH environment variable so that you do not need to enter its full path to run it. You add it to your path by adding the following to the `.profile` file in your home directory or the global `/etc/profile` file:

```
export PATH=$PATH:/opt/vertica/bin
```

Installing JDBC Driver on Linux, Solaris, AIX, and HP-UX

Note: The ODBC and JDBC client drivers are installed by the server `.rpm` files. If you have installed Vertica Analytics Platform on your Linux system for development or testing purposes, you do not need to download and install the client drivers on it—you just need to configure the drivers. To use ODBC, you need to create a DSN (see [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)). To use JDBC, you need to add the JDBC client driver to the Java CLASSPATH (see [Modifying the Java CLASSPATH](#)). (The JDBC client driver is not available on FIPS-compliant systems.)

Note: For additional details about supported platforms, see [Supported Platforms](#).

The JDBC driver is available for download from [myVertica portal](#). There is a single `.jar` file that works on all platforms and architectures. To download and install the file:

1. Open a Web browser and log in to [myVertica portal](#).
2. Click the Download tab and locate and download the JDBC driver.
3. You need to copy the `.jar` file you downloaded to a directory in your Java CLASSPATH on every client system with which you want to access Vertica. You can either:
 - Copy the `.jar` file to its own directory (such as `/opt/vertica/java/lib`) and then add that directory to your CLASSPATH (recommended). See [Modifying the Java CLASSPATH](#) for details.
 - Copy the `.jar` file to directory that is already in your CLASSPATH (for example, a directory where you have placed other `.jar` files on which your application depends).

Note: In the directory where you copied the `.jar` file, you should create a symbolic link named `vertica_jdk_5.jar` to the `.jar` file. You can reference this symbolic link anywhere you need to use the name of the JDBC library without having to worry any future upgrade invalidating the file name. This symbolic link is automatically created on server installs. On clients, you need to create and manually maintain this symbolic link yourself if you installed the driver manually. The [Installing the Client RPM on Red Hat and SUSE](#) create this link when they install the JDBC library.

Installing ODBC Drivers on Linux, Solaris, AIX, and HP-UX

Note: For additional details about supported platforms, see [Supported Platforms](#).

Read [Driver Prerequisites](#) before you proceed.

For Red Hat Enterprise Linux and SUSE Linux Enterprise Server, you can download and install a client RPM that installs both the ODBC and JDBC driver as well as the vsql client. See [Installing the Client RPM on Red Hat and SUSE](#).

Note: The ODBC and JDBC client drivers are installed by the server `.rpm` files. If you have installed Vertica Analytics Platform on your Linux system for development or testing purposes, you do not need to download and install the client drivers on it—you just need to configure the drivers. To use ODBC, you need to create a DSN (see [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)). To use JDBC, you need to add the JDBC client driver to the Java CLASSPATH (see [Modifying the Java CLASSPATH](#)). (The JDBC client driver is not available on FIPS-compliant systems.)

The ODBC driver installation packages are broken down by client platform on the [myVertica portal](#). The package's filename is named based on its operating system and architecture (for example, `vertica_8.1.xx_odbc_x86_64_linux.tar.gz`)

Important: Vertica provides the FIPS-compliant client driver only as an rpm for 64-bit clients. You can install this rpm only on FIPS-enabled machines. The FIPS client includes vsql and ODBC drivers. If you are installing the FIPS-specific client, refer to the section, [Installing the FIPS Client Driver for ODBC and vsql](#).

Installation Procedure

1. Open a Web browser and log in to [myVertica portal](#).
2. Click the Download tab and locate and download the driver package that corresponds to your client system.
3. If you did not directly download to the client system, transfer the downloaded file to it.
4. Log in to the client system as root.
5. If the directory `/opt/vertica/` does not exist, create it:

```
# mkdir -p /opt/vertica/
```

6. Copy the downloaded file to the `/opt/vertica/` directory. For example:

```
# cp vertica_8.1..xx_odbc_x86_64_linux.tar.gz /opt/vertica/
```

7. Change to the `/opt/vertica/` directory:

```
# cd /opt/vertica/
```

8. Uncompress the file you downloaded. For example:

```
$ tar vzxvf vertica_8.1..xx_odbc_x86_64_linux.tar.gz
```

Two folders are created: one for the include file, and one for the library files. The path of the library file depends on the processor architecture: `lib` for 32-bit libraries, and `lib64` for 64-bit libraries. So, a 64-bit driver client download creates the directories:

- `/opt/vertica/include`, which contains the header file
- `/opt/vertica/lib64`, which contains the library file

Post Driver Installation Configuration

You must configure the ODBC driver before you can use it. There are two required configuration files:

- The `odbc.ini` configuration file defines the Data Source Names (DSNs) that tell the ODBC how to access your Vertica databases. See [Creating an ODBC Data Source Name](#) for instructions to create this file.
- The `vertica.ini` configuration file defines some Vertica-specific settings required by the drivers. See [Required ODBC Driver Configuration Settings for Linux and UNIX](#) for instructions to create this file.

Note: If you are upgrading your ODBC driver, you must either update your DSNs to point to the newly-installed driver or create new DSNs. If your `odbc.ini` file references drivers defined in an `odbcinst.ini` file, you just need to update the `odbcinst.ini` file. See [Creating an ODBC Data Source Name \(DSN\)](#) for details.

Required ODBC Driver Configuration Settings for Linux and UNIX

In addition to DSN settings, Vertica provides additional ODBC client driver configuration parameters. These settings control the following:

- The text encoding used by the driver manager (for example, UTF-8 or UTF-16).
- The location of the directory containing the Vertica ODBC driver's error message files.
- Whether and how the ODBC driver logs messages.

On Linux and UNIX platforms, you must provide these additional settings manually so that the ODBC driver can function properly. To do so, edit the `vertica.ini` file to supply the necessary additional configuration settings. You specify where the ODBC driver can find the `vertica.ini` file using an environment variable named `VERTICAINI`. See [Required Settings](#).

Setting ODBC Driver Settings on Linux and UNIX-Like Platforms

Driver settings specific to Vertica are stored in a text file named `vertica.ini` (although you may choose a different file name). On Linux and UNIX platforms, you must edit the `vertica.ini` file to supply additional configuration settings before the ODBC driver can function properly. You tell the Vertica ODBC driver where to find the `vertica.ini` file using an environment variable named `VERTICAINI`.

Required Settings

On Linux and UNIX platforms, you must configure two settings in order for the ODBC driver to work correctly:

- `ErrorMessagesPath`
- `ODBCInstLib` (unless the driver manager's installation library is in a directory listed in the `LD_LIBRARY_PATH` or `LIB_PATH` environment variables).

If your driver manager does not use UTF-8 encoding, you need to set `DriverManagerEncoding` to the proper encoding.

Create a vertica.ini File

There is no standard location for the `vertica.ini` file—you can store the file anywhere that it is convenient for you on your client system. One possible location is in the `/etc` directory if you have multiple users on your client system that need to access it. You can also have a `vertica.ini` file in each user's home directory so users can alter their own settings. Wherever you store it, be sure users have read access to the file.

The format of the `vertica.ini` file is similar to the `odbc.ini` file, with a section followed by parameter definitions. Unlike the `odbc.ini` file, `vertica.ini` contains a single section named `Driver`:

```
[Driver]
```

Following the section definition, you add setting definitions, one per line. A setting definition consists of the setting name, followed by an equal sign (=), followed by the value. The value does not need quotes. For example, to set the `ODBCInstLib` setting, you add a line like this:

```
ODBCInstLib=/usr/lib64/libodbcinst.so
```

See [Additional Parameter Settings](#) for a list of the additional settings.

Set the VERTICAINI Environment Variable

You must set an environment variable named VERTICAINI to the absolute path of the `vertica.ini` file. The Vertica ODBC driver uses this variable to find the settings.

Where you set this variable depends on whether users on your client system need to have separate `vertica.ini` files. If you want to have a single, system-wide `vertica.ini` file, you can add a command to set VERTICAINI in `/etc/profile` or some other system-wide environment file. For example:

```
export VERTICAINI=/etc/vertica.ini
```

If users need individual `vertica.ini` files, set VERTICAINI in their `~/.profile` or similar configuration file. For example:

```
export VERTICAINI=~/.vertica.ini
```

Example vertica.ini File

The following example `vertica.ini` file configures the ODBC driver to:

- use the 64-bit UNIXODBC driver manager.
- get its error messages from the standard Vertica 64-bit ODBC driver installation directory.
- log all warnings and more severe messages to log files stored in the temporary directory.

```
[Driver]
DriverManagerEncoding=UTF-16
ODBCInstLib=/usr/lib64/libodbcinst.so
ErrorMessagesPath=/opt/vertica
LogLevel=4
LogPath=/tmp
```

Installing the FIPS Client Drivers

This topic details how to install the FIPS client drivers for JDBC and ODBC.

Installing the FIPS Client Driver for ODBC and vsql

Vertica offers a FIPS client for FIPS-compatible systems. A FIPS-compatible system is FIPS-enabled and includes the OpenSSL libraries.

The FIPS client supports ODBC and `vsq` and is offered in 64-bit only.

Prerequisites

Verify that your host system is running Red Hat Enterprise Linux 6.6.

The FIPS client installer checks your host system for the value of the `sysctl` parameter, `crypto.fips_enabled`. You must set this parameter to 1 (enabled). If your host is not enabled, the client does not install.

For other prerequisites, related specifically to ODBC, see [ODBC Prerequisites](#).

Installing the FIPS Client

To install the FIPS client driver package:

1. Download the FIPS client package from the [Vertica Marketplace](#) or from the [myVertica portal](#).
2. Log in to the client system as root.
3. Install the RPM package you downloaded:

```
# rpm -Uvh package_name.rpm
```

For ODBC, once you have installed the client package, you need to create a DSN and set some additional configuration parameters. For more information, see:

- [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)
- [Required ODBC Driver Configuration Settings for Linux and UNIX](#)

You may also want to add the vsql client to your PATH environment variable so that you do not need to enter its full path to run it. To do so, add the following to the `.profile` file in your home directory or the global `/etc/profile` file:

```
export PATH=$PATH:/opt/vertica/bin
```

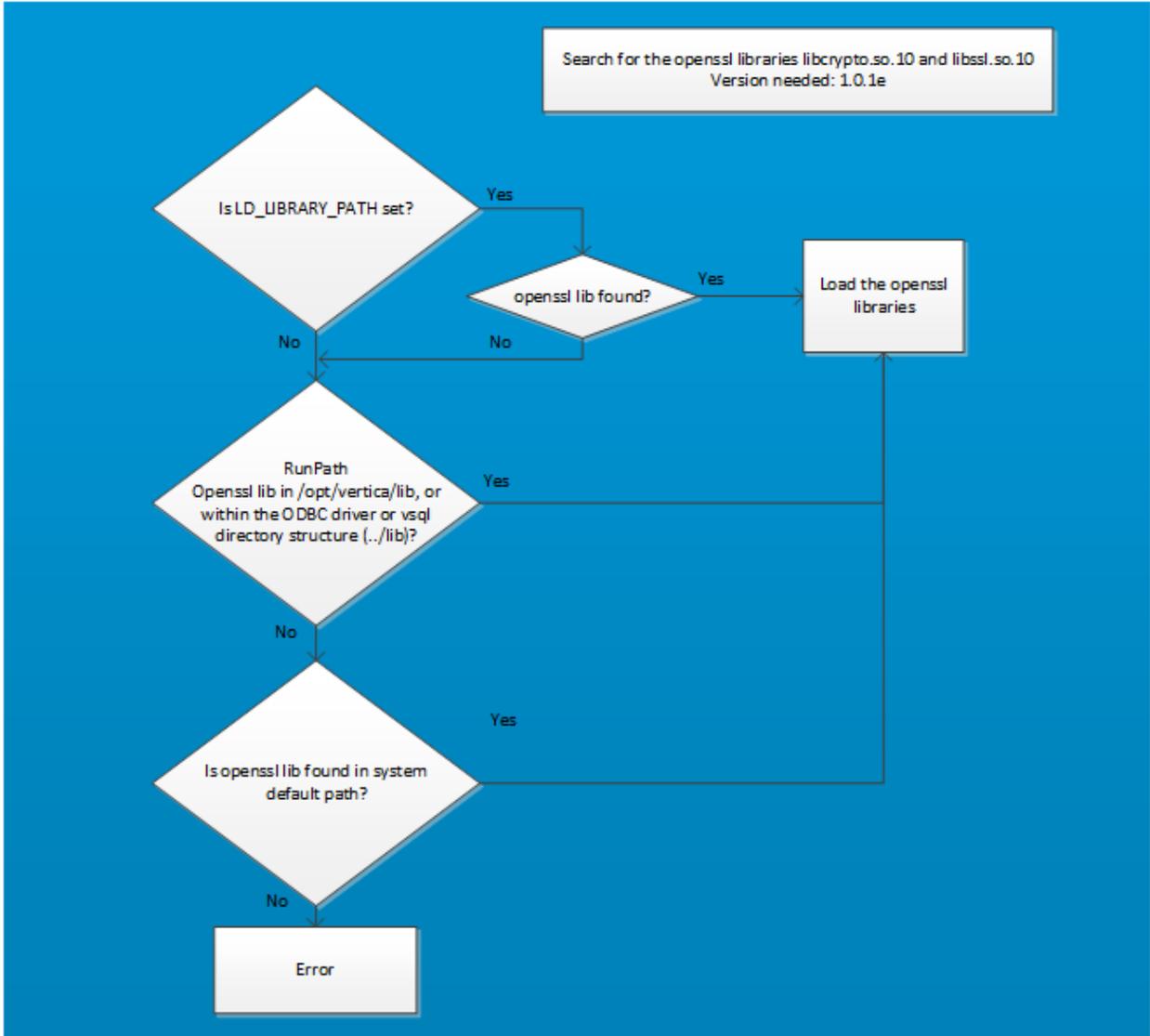
Client Searches for OpenSSL Libraries

When you launch the client application to connect to the server, the client searches for and loads the OpenSSL libraries `libcrypto.so.10` and `libssl.so.10` (version 1.0.1e):

- The client first checks to see if `LD_LIBRARY_PATH` is set.
- If the `LD_LIBRARY_PATH` location does not include the libraries, it checks `RunPath`, either `/opt/vertica/lib` or within the ODBC or vsql directory structure (`./lib`).

Important: The `LD_LIBRARY_PATH`, if set, directs the search path for the OpenSSL libraries. Be aware that the client loads the libraries from any set or preset `LD_LIBRARY_PATH` location.

The following figure depicts the search for the OpenSSL libraries.



Installing the FIPS Client Driver for JDBC

Vertica offers a JDBC client driver that is compliant with the Federal Information Processing Standard (FIPS). Use this JDBC client driver to access systems that are FIPS-compatible. For more information on FIPS in Vertica, see [Federal Information Processing Standard](#).

Implementing FIPS on a JDBC client requires a third-party JRE extension called [BouncyCastle](#), a collection of APIs used for cryptography. Use BouncyCastle APIs with JDK 1.7 and 1.8, and Red Hat 6.6.

Important: When using the JDBC FIPS-compliant client, expect some time lag for the client to connect efficiently and securely. If necessary, increase your system's entropy to ensure a fast and secure connection.

You need to add the FIPS BouncyCastle jar as the JVM JSSE provider, as follows:

1. Download the BouncyCastle FIPS jar file `bc-fips-1.0.0.jar` from the [BouncyCastle download page](#).
2. Add `bc-fips-1.0.0.jar` as a JRE library extension:

```
<path to jre>/lib/ext/bc-fips-1.0.0.jar
```

3. Add BouncyCastle as an SSL security provider in `<path to jre>/lib/security/java.security`:

```
security.provider.1=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
security.provider.2=com.sun.net.ssl.internal.ssl.Provider BCFIPS
security.provider.3=sun.security.provider.Sun
```

4. Use the following JVM `java -D` system property command arguments to set the KeyStore and TrustStore files to BCFIPS :

```
export JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.keyStoreProvider=BCFIPS
export JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStoreProvider=BCFIPS
```

For information on setting the SSL Keystore and Truststore, see [Configuring SSL for JDBC Clients](#).

5. Set the default type for the KeyStore implementation to BCFKS in `<path to jre>/lib/security/java.security`:

```
keystore type=BCFKS
ssl.keystore.type=BCFKS
```

Note: If you are using FIPS with BouncyCastle, you must create all client keys and certificates with the BCFKS store type, including the Vertica→Kafka key/certs.

6. On the command line, run the following command from <path to jre>/lib/ext to create the keystore and truststore. Make sure you use the BCFKS type:

```
$ <java bin path> keytool -keystore vertica.kafka.keystore.bcfks  
-storetype BCFKS  
-providertype BCFIPS  
-providerclass  
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider  
-provider  
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider  
-providerpath bc-fips-1.0.0.jar  
-alias CARoot  
-import -file (server.crt.der file path)
```

7. Enter the keystore password when prompted. The following message appears:

```
"Certificate was added to the keystore"
```

8. Run the java program with SSL DB:
 - a. Copy the vertica.kafka.keystore.bcfks keyStore from <path to jre>/lib/ext/ to the java program folder.

- b. Convert the Vertica server certificate to a form that java understands:

- i. openssl x509 -in server.crt -out server.crt.der -outform der

- ii. <java bin path>/keytool -keystore verticastore -keypasswd -storepass password -importkeystore -noprompt -alias verticasql -import -file server.crt.der

- c. Download the latest vertica jdbc driver from the [my.vertica.com download page](https://my.vertica.com/download).

9. After creation of verticastore, keyStore, and download jar, execute the following command to run java with debugging to test the implementation:

```
$ java -Djavax.net.debug=ssl -  
Djavax.net.ssl.keyStore='vertica.kafka.keystore.bcfks'  
-Djavax.net.ssl.keyStorePassword='password'  
-Djavax.net.ssl.trustStore='<path to verticastore>/verticastore'  
-Djavax.net.ssl.trustStorePassword='password'  
-cp .:vertica-jdbc-8.1.0-0.jar FIPSTest
```

Installing the Client Drivers and Tools on Windows

This section details how to install the client drivers and tools on Windows.

For connectivity through a JDBC connection, see [Installing the JDBC Client Driver for Windows](#).
For all other client drivers and tools, see [The Vertica Client Drivers and Tools for Windows](#).

Installing the JDBC Client Driver for Windows

To install the Vertica JDBC driver on your Windows client system, you must first download the cross-platform JDBC driver `.jar` file to your system. Then, choose the method that the Windows Java installation will use to find it.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the `.jar` file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an `.exe` file), which includes ODBC, vsqI, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 7.2.3 to 8.0.0 would not require that you uninstall before upgrade.

Download the JDBC Driver for Windows

1. On your Windows client system, open a browser, and log in to the [myVertica portal](#).
2. Install the Vertica JDBC driver for Windows:
 - a. Navigate to the Downloads tab, and scroll to the Client Software section.
 - b. Click the download link for the JDBC Driver for Windows installer.
3. Accept the license agreement, and wait for the download to complete.

Choose How Java Locates the JDBC Driver Library

For your Java client application to use the Vertica JDBC driver, the Java interpreter must be able to find the driver's library file. Choose one of these methods to tell the Java interpreter where to look for the library:

- Copy the JDBC .jar file you downloaded to the system-wide Java Extensions folder (C:\Windows\Sun\Java).
- Add the directory containing the JDBC .jar file to the CLASSPATH environment variable (see [Modifying the Java CLASSPATH](#)).
- Specify the directory containing the JDBC .jar using the -cp argument in the Java command line you use to start your Java command line.

The Vertica Client Drivers and Tools for Windows

You can obtain the Vertica Client Drivers and Tools for Windows installer through the [myVertica](#) portal. You can run the installer on either a 32-bit or 64-bit system, either as a [regular Windows installer](#) or [silently](#).

Components

The Vertica Client Drivers and Tools for Windows installs the following components on your system:

- [The ODBC Client Driver for Windows](#)
- [The OLE DB Client Driver for Windows](#)
- [The vsql Client for Windows](#)
- [The ADO.NET Driver for Windows](#)
- [The Microsoft Connectivity Pack for Windows](#)
- [The Visual Studio Plug-in for Windows](#)

Read [Fully Update Your System](#) before you proceed.

System Prerequisites

The Vertica Client Drivers and Tools for Windows released with Vertica Release 7.2. or later has basic system prerequisite requirements. The pack also requires that specific Microsoft components be installed for full integration.

For a list of all prerequisites, see the Supported Platforms document located at <http://my.vertica.com/docs>.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the .jar file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an .exe file), which includes ODBC, vsql, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 7.2.3 to 8.0.0 would not require that you uninstall before upgrade.

Fully Update Your System

Before you install the Vertica driver package, verify that your system is fully up to date with all Windows updates and patches. See the documentation for your version of Windows for instructions on how to run Windows update. The Vertica client libraries and vsql executable install updated Windows libraries that depend on Windows service packs. Be sure to resolve any issues that block the installation of Windows updates.

If your system is not fully up-to-date, you may receive error messages about missing libraries such as `api-ms-win-crt-runtime-l1-1-0.dll` when starting vsql.

.NET Framework

The .NET framework is not bundled into the Vertica Client Drivers and Tools for Windows. However, during installation, a web installer launches if Microsoft .NET 3.5 SP1 is not detected on your system. You then have the opportunity to download the framework. Also, if your operating system version includes .NET 3.5 SP1, but it is not turned on, the installer turns on the feature.

If you have Visual Studio 2010 or 2012 installed, your system already includes Microsoft .NET Framework 4.0 or 4.5, respectively. You also need Microsoft .NET 3.5 SP1 to use the Vertica Client Drivers and Tools for Windows integration features.

Use the following links to download the appropriate version of .NET framework directly from Microsoft:

- For .NET Framework 3.5 SP1:

<http://www.microsoft.com/en-us/download/details.aspx?id=22>

- For .NET Framework 4.0:

<http://www.microsoft.com/en-us/download/details.aspx?id=17851>

- For .NET Framework 4.5:

<http://www.microsoft.com/en-us/download/details.aspx?id=17851>

Microsoft Visual Studio

The Vertica Client Drivers and Tools for Windows installer provides a Visual Studio plug-in which allows you to use Vertica as a Visual Studio Data Source for Visual Studio 2008, 2010, 2012, 2013, or 2015. The connection properties for the plug-in are the same as [ADO.NET Connection Properties](#).

Important: You must have Visual Studio and the matching SDK installed to use the Visual Studio plug-in.

After installing the plug-in, you can use it to access your Vertica database from within Visual Studio. If you do not have the SDK installed, download the SDK specific to your version of Visual Studio.

Note: For Visual Studio 2015, you do not need to download the SDK separately as it is included as an installation option with the Visual Studio installation. For more information, refer to the [Microsoft documentation](#).

- For the Microsoft Visual Studio 2008 SDK:
<http://www.microsoft.com/en-us/download/details.aspx?id=508>
- For the Microsoft Visual Studio 2008 SP1 SDK:
<http://www.microsoft.com/en-us/download/details.aspx?id=21827>
- For the Microsoft Visual Studio 2010 SDK:
<http://www.microsoft.com/en-us/download/details.aspx?id=2680>
- For the Microsoft Visual Studio 2010 SP1 SDK:
<http://www.microsoft.com/en-us/download/details.aspx?id=21835>
- For the Microsoft Visual Studio 2012 SDK:
<http://www.microsoft.com/en-us/download/details.aspx?id=30668>
- For the Microsoft Visual Studio 2013 SDK:
<http://www.microsoft.com/en-us/download/details.aspx?id=40758>

If the Microsoft Visual Studio SDK is missing when you begin the installation, a dialog box opens to tell you so. You can choose to ignore this dialog box.

Configuring BIDS or SSDT-BI Integration

The Vertica Client Drivers and Tools for Windows installer provides BIDS (Visual Studio 2008) or SSDT-BI (Visual Studio 2010, 2012, 2013, or 2015) integration. To use BIDS or SSDT-BI, follow this process:

1. Install the BIDS or SSDT-BI development tool add-on for Visual Studio.
2. Verify that SQL Server is installed on the same or a different machine.
3. Verify that the SQL Server Shared Features for BIDS or SSDT-BI have been activated.

You can then develop packages using BIDS or SSDT-BI, creating your projects using SQL Server's SSIS, SSAS, SSRS features. To use these features, you must connect to Vertica through the Vertica ADO.NET driver (for SSIS and SSRS) or the OLE DB driver (for SSAS).

For more information, see [Microsoft Components](#).

Microsoft SQL Server

Use SQL Server 2008, 2012, 2014 or 2016. The Vertica Client Drivers and Tools for Windows installer enables support for the following:

- **SQL Server 2008, 2012, 2014, and 2016:**
 - SQL Server Integration Services (SSIS)
 - SQL Server Reporting Services (SSRS)
 - SQL Server Analysis Services (SSAS)
- **SQL Server using Visual Studio 2008** — Business Intelligence Development Studio (BIDS)
- **SQL Server using Visual Studio 2010, 2012, 2013, and 2015** — SQL Server Data Tool - Business Intelligence (SSDT-BI)

Note: For SQL Server 2012, you can use either SQL Server 2012 or SQL Server 2012 SP1.

To use the enhanced Vertica .NET support, you must first install SQL Server. Then, you can install the Client Drivers and Tools for Windows. The following components must be installed on the SQL server:

For...	Install...
SSAS	The Analysis Services Instance Feature.
SSRS	The Reporting Services Instance Feature.
SSIS (Data Type Mappings)	The SQL Server Integration Services Shared Feature.
BIDS (for Visual Studio 2008)	Business Intelligence Development Studio Shared Feature only <i>after</i> installing Microsoft Visual Studio 2008.
SSDT-BI (Visual Studio 2010, 2012, 2013, or 2015)	SQL Server Data Tool - Business Intelligence Shared Feature only <i>after</i> installing Microsoft Visual Studio 2010, 2012, 2013, or 2015.

Download the Client Drivers and Tools for Windows

1. On your Windows client system, open a browser, and log in to the [myVertica portal](#).
2. Install the Vertica Client Drivers and Tools for Windows:
 - a. Navigate to the Downloads tab, and scroll to the Client Software section. Click the link for the [Vertica Client Drivers](#) download page.
 - b. Click the download link for the Vertica Client Drivers and Tools installer.
3. Accept the license agreement, and wait for the download to complete.

Install or Upgrade the Client Drivers and Tools for Windows

As the Windows Administrator, double-click the installer to start the installation. Follow the prompts as the wizard guides you through each step of the process.

By default, the installer installs the client drivers and tools in C:\Program Files\Vertica Systems\. You have the option of changing this location during installation.

Note: The install wizard allows you to choose the components you want to install. Under your Windows Control Panel, the installer adds a program for each installed component. Among the programs the installer adds is a program named, Vertica Bootstrapper. You can right-click on the bootstrapper to modify, repair, or uninstall the Vertica client.

Important: The Vertica Microsoft Connectivity Pack is included as part of the Vertica Client Drivers and Tools for Windows. If you plan to use the Microsoft Connectivity Pack to access Microsoft Business Intelligence tools, reboot your system after installation to ensure integration.

Upgrading the Client Drivers and Tools for Windows

You do not need to uninstall the Client Drivers and Tools for Windows before upgrading. The installation program upgrades the existing drivers and tools in place.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the `.jar` file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an `.exe` file), which includes ODBC, vsql, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 7.2.3 to 8.0.0 would not require that you uninstall before upgrade.

Silently Install or Upgrade the Client Drivers and Tools for Windows

1. As a Windows Administrator, open a command-line session, and change directory to the folder that contains the installer.
2. Run the command:

```
VerticaSetup.exe -q -install InstallFolder="C:\Program Files\Vertica Systems"
```

The client drivers and tools are silently installed in `C:\Program Files\Vertica Systems\`.

Upgrading the Client Drivers and Tools for Windows

You do not need to uninstall the Client Drivers and Tools for Windows before upgrading. The installation program upgrades the existing drivers and tools in place.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the .jar file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an .exe file), which includes ODBC, vsql, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 7.2.3 to 8.0.0 would not require that you uninstall before upgrade.

Post-Installation Steps for ODBC Driver and vsql Client

After you install the Vertica Client Drivers and Tools for Windows, there are additional steps you must take for the ODBC driver and vsql client to function correctly.

- **ODBC Driver** — After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see [Creating an ODBC DSN for Windows Clients](#).
- **vsql Client** — The vsql client does not have a shortcut. Before you can start using vsql, you must add the vsql executable to the Windows PATH environment variable. The method for altering the PATH environment variable depends on the version of the Microsoft Windows operating system you are running. To start vsql and show the help list, open a command window, and type `vsq1 -?` at the command prompt. See [Using vsql for Windows Users](#) for important details about using vsql in a Windows console.

Uninstalling, Modifying, or Repairing the Client Drivers and Tools

To uninstall, modify, or repair the client drivers and tools, run the Client Drivers and Tools for Windows installer.

The installer provides three options:

Action	Description
Modify	Remove installed client drivers and tools or install missing client drivers and tools.
Repair	Reinstall already-installed client drivers and tools.
Uninstall	Uninstall all of the client drivers and tools.

Silently Uninstall the Client Drivers and Tools

1. As a Windows Administrator, open a command-line session, and change directory to the folder that contains the installer.
2. Run the command:

```
VerticaSetup.exe -q -uninstall
```

The client drivers and tools are silently uninstalled.

Components of the Client Drivers and Tools on Windows

The following sections describe the components in the Client Drivers and Tools for Windows in more detail:

- [The ODBC Client Driver for Windows](#)
- [The vsql Client for Windows](#)
- [The Microsoft Connectivity Pack for Windows](#)
- [The OLE DB Client Driver for Windows](#)
- [The ADO.NET Driver for Windows](#)
- [The Visual Studio Plug-in for Windows](#)

The ODBC Client Driver for Windows

The Vertica ODBC driver for Windows is installed as part of the Client Drivers and Tools for Windows.

After Installing the ODBC Driver

After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see [Creating an ODBC DSN for Windows Clients](#).

ODBC Driver Settings on Windows

ODBC driver settings are automatically configured using the Vertica Client Drivers and Tools installer on Windows. The values for the settings are stored in the Windows registry under the path `HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\ODBC\Driver`. It is not necessary to configure additional ODBC driver settings on Windows platforms beyond what is automatically configured by the installer. You can, however, set the ODBC driver settings using the Windows ODBC Data Source Configuration window.

See [Additional Parameter Settings](#) for a list of additional settings for the ODBC client driver. See [Register the ODBC Driver as a Windows Event Log Provider, and Enable the Logs](#) for information on how to send ODBC log entries to Event Tracing for Windows (ETW).

Diverting ODBC Log Entries to ETW

On Windows clients, you can direct Vertica to send ODBC log entries to Event Tracing for Windows (ETW). Once set, ODBC log entries appear in the Windows Event Viewer. To use ETW:

- Register the driver as a Windows Event Log provider, and enable the logs.
- Activate ETW by adding a string value to your Windows Registry.
- Understand how Vertica compresses log levels for the Windows Event Viewer.
- Know where to find the logs within Event Viewer.
- Understand the meaning of the Event IDs in your log entries.

Register the ODBC Driver as a Windows Event Log Provider, and Enable the Logs

To use ETW logging, you must register the ODBC driver as a Windows Event Log provider. You can choose to register either the 32-bit or 64-bit driver. Once you have registered the driver, you must enable the logs.

Important: If you do not both register the driver and enable the logs, output is directed to stdout.

1. Open a command prompt window as Administrator, or launch the command prompt with the Run as Administrator option.

Important: You must have administrator privileges to successfully complete the next step.

2. Run the command `wevtutil im` to register either the 32-bit or 64-bit version of the driver.
 - a. For the 64-bit ODBC driver, run:

```
wevtutil im "c:\Program Files\Vertica Systems\ODBC64\lib\VerticaODBC64.man"  
/resourceFilePath:"c:\Program Files\Vertica Systems\ODBC64\lib\vertica_8.1_odbc_3.5.dll"  
/messageFilePath:"c:\Program Files\Vertica Systems\ODBC64\lib\vertica_8.1_odbc_3.5.dll"
```

- b. For the 32-bit ODBC driver, run:

```
wevtutil im "c:\Program Files (x86)\Vertica Systems\ODBC32\lib\VerticaODBC32.man"  
/resourceFilePath:"c:\Program Files (x86)\Vertica Systems\ODBC32\lib\vertica_8.1_odbc_  
3.5.dll"  
/messageFilePath:"c:\Program Files (x86)\Vertica Systems\ODBC32\lib\vertica_8.1_odbc_3.5.dll"
```

3. Run the command `wevtutil sl` to enable the logs.
 - a. For 64-bit ODBC driver logs, run:

```
wevtutil sl VerticaODBC64/e:true
```

- b. For the 32-bit ODBC driver logs, run:

```
wevtutil sl VerticaODBC32/e:true
```

Note: Should you want to later disable the logs, you can use the same `wevtutil sl` command, substituting `/e:false` in place of `/e:true` when you issue the statement. Alternatively, you can enable or disable logs within the Windows Event Viewer itself.

Add the String Value LogType

By default, Vertica does not send ODBC log entries to ETW. To activate ETW, add the string LogType to your Windows registry, and set its value to ETW.

1. Start the registry editor by typing `regedit.exe` in the Windows Run command box.
2. Navigate to the correct location in the registry.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\ODBC\Driver
```

3. Right-click in the right pane of the **Registry Editor** window. Select **New** and then select **String Value**.
4. Change the name of the string value from `New Value #1` to `LogType`.
5. Double-click the new `LogType` entry. When prompted for a new value, enter `ETW`.
6. Exit the registry editor.

ETW is off by default. When ETW is activated, you can subsequently turn it off by clearing the value `ETW` from the `LogType` string.

Event Viewer Log Levels

The LogLevel parameter setting is described in the section, [Additional Parameter Settings](#). The parameter allows you to specify a LogLevel of 0 through 6. Be aware that Vertica compresses the log levels for the Windows Event Viewer. The six levels are compressed to four in Event Viewer.

Vertica LogLevel Setting	Vertica LogLevel Description	Entries are sent to Event Viewer as log level...	Event Viewer Displays...
0	(No logging)	0	(No logging)
1	Fatal Errors	1	Critical
2	Errors	2	Error
3	Warnings	3	Warning
4	Info	4	Information
5	Debug	4	
6	Trace (all messages)	4	

Examples:

- A LogLevel setting of 5 sends fatal errors, errors, warnings, info and debug log level entries to Event Viewer as Level 4 (Information).
- A LogLevel setting of 6 sends fatal errors, errors, warnings, debug and trace log level entries to Event Viewer as Level 4.

Where to Find Logs in Event Viewer

1. Launch the **Windows Event Viewer**.
2. From **Event Viewer (Local)**, expand **Applications and Services Logs**.
3. Expand the folder that contains the log you want to review (for example, `VerticaODBC64`).
4. Select the Vertica ODBC log under the folder. Entries appear in the right pane.

Event Log Entry: Event ID

Once you have chosen an ODBC log in Event Viewer, note the value in the **Event ID** field.

Each Event Log entry includes one of four Event IDs. An Event ID of 0 is informational (debug, info, and trace events), 1 is an error, 2 is a fatal event, and 3 is a warning.

The vsql Client for Windows

The Vertica vsql client for Windows is installed as part of the Client Drivers and Tools for Windows.

There is no shortcut for the vsql client. Before you can start using vsql, you must add the vsql executable to the Windows PATH environment variable. The method for altering the PATH environment variable depends on the version of the Microsoft Windows operating system you are running. After you have made the change to your PATH environment variable, start a command window and type `vsql -?` at the command prompt to start vsql and show the help list.

See [Using vsql for Windows Users](#) for important details about using vsql in a Windows console.

Using vsql for Windows Users

Font

The default raster font does not work well with the ANSI code page. Set the console font to "Lucida Console."

Console Encoding

vsq1 is built as a "console application." The Windows console windows use a different encoding than the rest of the system, so take care when you use 8-bit characters within vsq1. If vsq1 detects a problematic console code page, it warns you at startup.

To change the console code page, set the code page by entering `cmd.exe /c chcp 1252`.

Note: 1252 is a code page that is appropriate for European languages. Replace it with your preferred locale code page.

Running Under Cygwin

Verify that your `cygwin.bat` file does not include the "tty" flag. If the "tty" flag is included in your `cygwin.bat` file, then banners and prompts are not displayed in `vsq`.

To verify, enter:

```
set CYGWIN=binmode tty ntsec
```

To remove the "tty" flag, enter:

```
set CYGWIN=binmode ntsec
```

Additionally, when running under Cygwin, `vsq` uses Cygwin shell conventions as opposed to Windows console conventions.

Tab Completion

Tab completion is a function of the shell, not vsql. Because of this, tab completion does not work the same way in Windows vsql as it does on Linux versions of vsql.

On Windows, instead of using tab-completion, press F7 to pop-up a history window of commands. You can also press F8 after typing a few letters of a command to cycle through commands in the history buffer which begin with the same letters.

The Microsoft Connectivity Pack for Windows

The Vertica Microsoft Connectivity Pack for Windows provides a configuration file for you to access Microsoft Business Intelligence tools. The Connectivity Pack is installed as part of the Client Drivers and Tools for Windows.

To learn about which Microsoft components are configured with the Microsoft Connectivity Pack, see [Microsoft Components](#).

Microsoft Components

This section describes the Microsoft Business Intelligence components you can use with Microsoft Visual Studio and Microsoft SQL Server. After configuration, you can use these Microsoft components to develop business solutions using your Vertica server.

Important: The Vertica Microsoft Connectivity Pack is included as part of the Vertica Client Drivers and Tools for Windows. If you plan to use the Microsoft Connectivity Pack to access Microsoft Business Intelligence tools, reboot your system after installation to ensure integration.

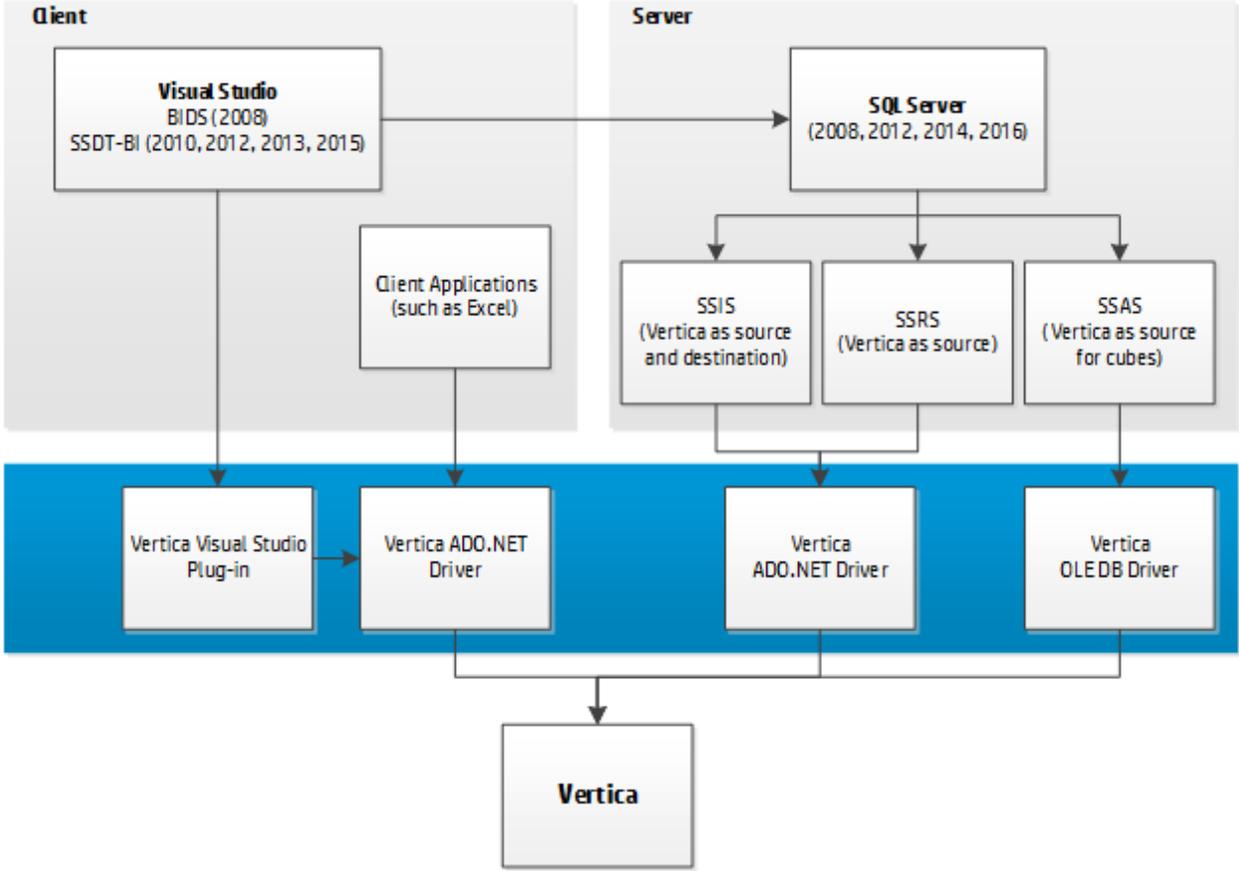
Microsoft Component Configuration

The Vertica ADO.NET driver, the Visual Studio plug-in, and the OLE DB driver allow you to integrate your Vertica server with an environment that includes Microsoft components previously installed on your system. Additional tools are also available for integration with Microsoft SQL Server.

The available drivers provide integration with the following Microsoft components:

- Business Intelligence Development Studio (BIDS) for Visual Studio 2008 for use with SQL Server 2012. BIDS is a client-based application used to develop business intelligence solutions based on the Microsoft Visual Studio development environment. It includes additional project types specific to SQL Server Business Intelligence. As a developer, you can use BIDS to develop business solutions.
- SQL Server Data Tool - Business Intelligence (SSDT-BI) for Visual Studio 2010/2012/2013/2015 for use with SQL Server 2012, 2014, and 2016. SSDT-BI replaces BIDS for Visual Studio 2010, 2012, 2013, and 2015. It serves the same purpose as BIDS, providing a development environment for developing business intelligence solutions.
- SQL Server Analysis Services (SSAS) for SQL Server 2008, 2012, 2014, and 2016. Use SSAS for OLAP and data mining, while using Vertica as the source for cube creation.
- SQL Server Integration Services (SSIS) for SQL Server 2008, 2012, 2014, and 2016. SSIS provides SQL Server Type Mappings to map data types between Vertica and SQL Server. Use SSIS for data migration, data integration and workflow, and ETL.

The following figure displays the relationship between Microsoft components and Vertica dependencies.



BIDS and SSDT-BI

Business Intelligence Development Studio (BIDS) is available in Microsoft Visual Studio 2008 with additional project types that are specific to SQL Server business intelligence. BIDS is the primary environment that you use to develop business solutions that include Analysis Services, Integration Services, and Reporting Services projects.

SQL Server Data Tool - Business Intelligence (SSDT-BI) replaces BIDS for Visual Studio 2010, 2012, 2013, and 2015. It serves the same purpose as BIDS, providing a development environment for developing business solutions.

Both BIDS and SSDT-BI are client-based applications that include additional project types specific to SQL Server Business Intelligence.

You can use the Visual Studio Shell Integration plug-in to browse a database from within the Visual Studio Server Explorer. This capability allows you to work outside of BIDS or SSDT-BI development to perform tasks, such as listing tables or inserting data. When you use Visual Studio in BIDS or SSDT-BI mode, you can develop business solutions using the data in your Vertica database. For example, you can create cubes or open tables.

Microsoft does not support the following configurations:

- You cannot use Microsoft Visual Studio 2008 with BIDS development to create a SQL Server 2012 Business Intelligence solution.
- You cannot use Microsoft Visual Studio 2010/2012/2013/2015 with SSDT-BI development to create a SQL Server 2008 Business Intelligence solution.

SQL Server Analysis Services (SSAS) Support

BIDS or SSDT-BI includes the Analysis Services project for developing online analytical processing (OLAP) for business intelligence applications. This project type includes templates for:

- Cubes
- Dimensions
- Data sources
- Data source views

It also provides the tools for working with these objects.

Note: Micro Focus recommends that you use the Vertica OLE DB driver when connecting to the Vertica server from SSAS due to improved performance.

You can find the OLE DB connection properties in [OLE DB Connection Properties](#).

SQL Server Integration Services (SSIS) Support

BIDS or SSDT-BI includes the Integration Services project for developing ETL solutions. This project type includes templates for:

- Packages
- Data sources
- Data source views

It also provides the tools for working with these objects.

You can find support for using Vertica as a data source and target from both SSIS and the import/export wizard. You must install mapping files specific to Vertica on the Integration Server and BIDS or SSDT-BI workstation to enable this capability. The Vertica Client Drivers and Tools for Windows installs these mapping files as the "SQL Server Type Mappings" component (s) in both 32-bit and 64-bit versions.

Note: Always use the Vertica ADO.NET driver when connecting to the Vertica server from SSIS.

SQL Server Reporting Services (SSRS) Support

BIDS or SSDT-BI includes Report projects for developing reporting solutions.

You can use Vertica as a data source for Reporting Services. The installer implements various configuration file modifications to enable this capability on both the BIDS or SSDT-BI workstation and the Reporting Services server.

Compatibility Issues and Limitations

This section lists compatibility issues and limitations for integrating the Microsoft Connectivity Pack with Microsoft Visual Studio and Microsoft SQL Server.

BIDS and SSDT-BI Limitations

BIDS and SSDT-BI are 32-bit development environments for Analysis Services, Integration Services, and Reporting Services projects. They are not designed to run on the Itanium 64-bit architecture and thus are not installed on Itanium servers.

SSAS Limitations

- The SSAS Tabular Model is not supported.
- If, after installing the Vertica OLE DB driver, an SSAS cube build fails, restart the SSAS service.

SSIS Data Type Limitations

The following sections cover data type limitations when using SQL Server Integration Services (SSIS).

Time Data Transfer

When transferring time data, SSIS uses the TimeSpan data type that supports precision greater than six digits. The Vertica ADO.NET driver translates TimeSpan as an Interval data type that supports up to six digits. The Interval type is not converted to the TimeSpan type during transfer. As a result, if the time value has a precision of more than six digits, the data is truncated, not rounded.

For information on ADO.NET data types, refer to [ADO.NET Data Types](#).

DATE and DATETIME Precision

To function without errors, DATE and DATETIME have a range from 0001-01-01 00:00:00.0000000 to 9999-12-31 23:59:59.9999999.

In SSIS, the DATETIME type (DT_TIMESTAMP) supports only a scale up to three decimal places for seconds. Any decimal places after that are automatically discarded. You can perform derived column transformations only on DATETIME values between January 1, 1753 through December 31, 9999.

Numeric Precision

The maximum and minimum decimal allowed is:

- Max: +79,228,162,514,264,337,593,543,950,335
- Min: -79,228,162,514,264,337,593,543,950,335

For example, if the scale is 16, the range of values is:

+/- 7,922,816,251,426.4337593543950335

The valid scale range is any number that is smaller than 29 and greater than 38. Using a scale between 29 and 38 does not generate an error.

See: <http://msdn.microsoft.com/en-us/library/system.decimal.maxvalue.aspx>

Unsupported Floating Point Values

SQL Server does not support NaN, Infinity, or –Infinity values. These values are supported when you use SSIS to transfer between Vertica instances, but they are not supported with a SQL Server Destination.

String Conversion

The CHAR and VARCHAR data types used in SSIS are DT_WSTR, with a maximum length of 4000 characters.

In SSIS, Vertica strings are converted to Unicode strings in SSIS to handle multi-lingual data. You can convert these strings to ASCII using a Data Conversion Task.

Scale

Whenever you use a scale greater than 38, SSIS replaces it with a value of 4.

Interval Conversion

SSIS does not support interval types. It converts them to TIME and strips off the day component. Any package that has interval types greater than a day returns incorrect results.

Data Mapping Issues with SQL Server Import and Export Wizard

When you create an Integrated Services package (SSIS) using the SQL Server Import and Export Wizard, certain data types do not automatically map correctly. A mapping issue can occur when you use the wizard with:

- SQL Server Native OLE DB Provider for SQL Server 2008 or 2012
- SQL Server Native Client 10.0/11.0 Provider for SQL Server 2010/2012

To avoid this issue, manually change the type mappings with BIDS or SSDT-BI.

Data Transfer Failures

When using an Integrated Services package (SSIS) with the SQL Server OLE DB Provider for SQL Server 2008 or 2012, certain data type transfers can fail when transferring from Vertica to SQL Server. To avoid this issue, use either BIDS or SSDT-BI when transferring data.

Batch Insert of VARBINARY/LONG VARBINARY Data Types

Sometimes, one row of a batch insert of VARBINARY or LONG VARBINARY data types exceeds the data type limit:

- VARBINARY: 65 KB
- LONG VARBINARY: 32 MB

In such cases, all rows are rejected, rather than just the one row whose length exceeds the type limit. The batch insert fails with the message "row(s) are rejected".

To avoid this issue, use a predicate to filter out rows from the source that do not fit into the receiving database.

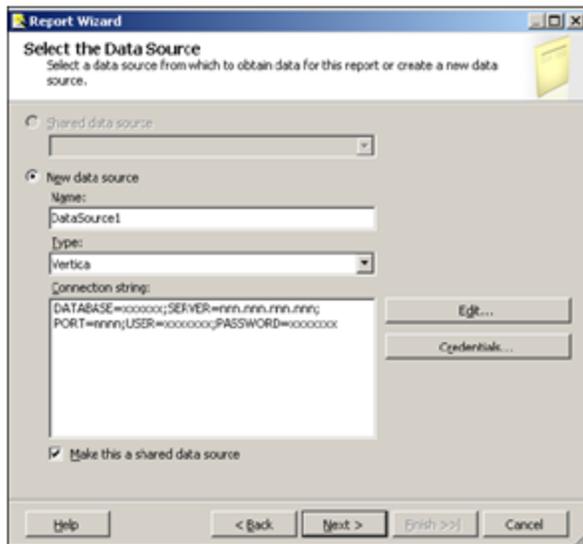
Boolean Queries in SQL Server Query Designer

When issuing a Boolean query in SQL Server Query Designer, you must enclose Boolean column values in quotes. Otherwise, you receive a SQL execution error (for example, `someboolean = 'true'`).

SSRS Limitations

Data Connection Wizard Workaround

The SSRS Report Wizard provides a data connection wizard. After you select the wizard and enter all the connection information, the **OK** button is disabled. You cannot save your work and continue. The workaround is to not use the wizard and to use the following panel instead:



Report Wizard - Query Designer

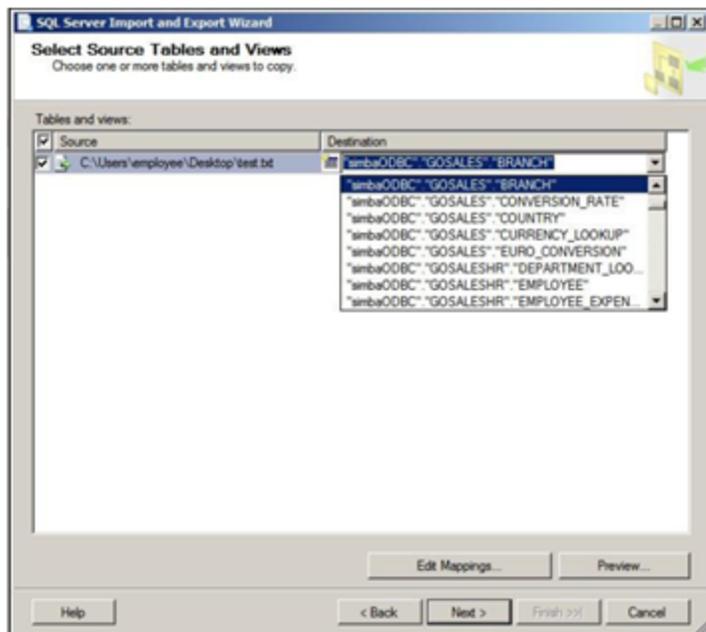
Vertica uses the Report Wizard's Generic Query Designer. Other data sources use a Graphical Query Designer that supports building queries visually. The Graphical Query Designer is a part of a package called Visual Data Tools (VDT). The Graphical Query Designer works only with Generic OLE DB providers and the built-in providers. You cannot use it with the Vertica Data Provider.

Report Builder

Report Builder is a web-based report design tool. It does not support creating reports using custom data extensions, so you cannot use it with Vertica. When you create a report using Report Builder, existing Vertica data sources appear in the list of available data sources. However, choosing an Vertica data source causes an error.

Schema Name Not Automatically Provided when Mapping Vertica Destination

Currently, when you map a Vertica destination, the schema name is not automatically provided. You must enter it manually or pick it from the drop-down menu as follows:



The OLE DB Client Driver for Windows

The Vertica OLE DB driver for Windows is installed as part of the Client Drivers and Tools for Windows. The values for the OLE DB driver's settings are stored in the Windows registry under the path HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\OLEDB\Driver.

For information on how the OLE DB driver integrates with Microsoft components previously installed on your system, see [Microsoft Component Configuration](#).

OLE DB Connection Properties

Use the **Connection Manager** to set the OLE DB connection string properties, which define your connection. You access the **Connection Manager** from within Visual Studio.

These connection parameters appear on the **Connection** page.

Parameters	Action	Default Value
Provider	Select the native OLE DB provider for the connection.	None
OLE DB Provider	Indicates Vertica OLE DB Provider.	None
Server or file name	Enter the server or file name.	None

Parameters	Action	Default Value
Location	Not supported.	Disabled.
Use Windows NT Integrated Security	Not supported.	Disabled.
Use a specific user name and password	Enter a user name and password. Connect with No Password: Select the Blank password check box. Save and Encrypt Password: Select Allow saving password .	None
Initial Catalog	The name of the database running on the server.	None

The **All** page from the **Connection Manager** dialog box includes all possible connection string properties for the provider.

The table that follows lists the connection parameters for the **All** page.

For OLE DB properties information specific to Microsoft, see the Microsoft documentation [OLE DB Properties](#).

Parameters	Action	Default Value
Extended Properties	Not supported.	Leave blank. Do not set this field.
Locale Identifier	Indicates the Locale ID.	0
Mode	Specifies access permissions.	0
Connect Timeout	Not supported. The value can be set, but has no effect.	0
General Timeout	Not supported. The value can be set, but has no effect.	0
File Name	Not supported. The value can be	Blank

Parameters	Action	Default Value
	set, but has no effect.	
OLE DB Services	Specifies which OLE DB services to enable or disable.	Default
Password	Specifies the password for the User ID. For no password, insert an empty string.	None (If no password specified, login succeeds only if the user has not set a password.)
Persist Security Info	A security measure. When False, security sensitive-information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state.	True
User ID	The database username.	None
Data Source	The host name or IP address of any active node in a Vertica cluster. You can provide an IPv4 address, IPv6 address, or host name. In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.	None
Initial Catalog	The name of the database running on the server.	None
Provider	The name of the OLE DB Provider to use when connecting to the Data Source.	VerticaOLEDB.1

Parameters	Action	Default Value
BackupServerNode	<p>A designated host name or IP address to use if the ServerName host is unavailable. Enter as a string.</p> <p>Connection attempts continue until successful or until the list of server nodes is exhausted.</p> <p>Valid values: Comma-separated list of servers optionally followed by a colon and port number. For example: server1:5033, server2:5034</p>	None
ConnectionLoadBalance	<p>A Boolean value that determines whether the connection can be redirected to a host in the database other than the ServerNode.</p> <p>This parameter affects the connection only if load balancing is set to a value other than NONE. When the node differs from the node that the client is connected to, the client disconnects and reconnects to the targeted node. See About Native Connection Load Balancing in the Administration Guide.</p>	False
ConnSettings	<p>SQL commands that the driver should execute immediately after connecting to the server. Use to configure the connection, such as setting a schema search path.</p> <p>Reserved symbol:',' To set multiple parameters in this field use '%3B' for ','.</p> <p>Spaces: Use '+'.</p>	None

Parameters	Action	Default Value
ConvertSquareBracketIdentifiers	Controls whether square-bracket query identifiers are converted to a double quote identifier for compatibility when making queries to a Vertica database.	False
DirectBatchInsert	<p>Controls where data inserted through the connection is stored.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • True — Data inserted directly into ROS containers. • False — Stores data using AUTO mode. <p>When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</p>	False
KerberosHostName	Provides the instance or host name portion of the Vertica Kerberos principal; for example: vertica/ host @EXAMPLE.COM	None.
KerberosServiceName	Provides the service name portion of the Vertica Kerberos principal; for example: vertica / host @EXAMPLE.COM	None
Label	Sets a label for the connection on the server. This value appears in the session_id column of the V_MONITOR.SESIONS system table.	None
LogLevel	Specifies the amount of information included in the log.	None

Parameters	Action	Default Value
	Leave this field blank or set to 0 unless otherwise instructed by Vertica Customer Support.	
LogPath	The path for the log file.	None
Port	The port number on which Vertica listens for OLE DB connections.	None (If not set, uses port 5433.)
PreferredAddressFamily	The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are: <ul style="list-style-type: none"> • ipv4—Connect to the server using IPv4. • ipv6—Connect to the server using IPv6. • none—Use the IP address provided by the DNS server. 	None
SSLCertFile	The absolute path of the client's public certificate file. This file can reside anywhere on the system.	None
SSLKeyFile	The absolute path to the client's private key file. This file can reside anywhere on the system.	None
SSLMode	Controls whether the connection to the database uses SSL encryption. Valid values follow. Valid Values: <ul style="list-style-type: none"> • require—Requires the server to use SSL. If the server cannot provide an encrypted channel, the connection fails. 	Prefer

Parameters	Action	Default Value
	<ul style="list-style-type: none">• prefer—Prefers that the server use SSL. If the server does not offer an encrypted channel, the client requests one. The first attempt is made with SSL. If that attempt fails, the second attempt is over a clear channel.• allow—Makes a connection to the server whether or not the server uses SSL. The first attempt is made over a clear channel. If that attempt fails, a second attempt is over SSL.• disable—Never connects to the server using SSL. Typically, you use this setting for troubleshooting.	

Diverting OLE DB Log Entries to ETW

On Windows clients, you can direct Vertica to send OLE DB log entries to Event Tracing for Windows (ETW). Once set, OLE DB log entries appear in the Windows Event Viewer. To use ETW:

- Register the driver as a Windows Event Log provider, and enable the logs.
- Activate ETW by adding a string value to your Windows Registry.
- Understand how Vertica compresses log levels for the Windows Event Viewer.
- Know where to find the logs within Event Viewer.
- Understand the meaning of the Event IDs in your log entries.

Register the OLE DB Driver as a Windows Event Log Provider, and Enable the Logs

To use ETW logging, you must register the OLE DB driver as a Windows Event Log provider. You can choose to register either the 32-bit or 64-bit driver. Once you have registered the driver, you must enable the logs.

Important: If you do not both register the driver and enable the logs, output is directed to stdout.

1. Open a command prompt window as Administrator, or launch the command prompt with the Run as Administrator option.

Important: You must have administrator privileges to successfully complete the next step.

2. Run the command `wevtutil im` to register either the 32-bit or 64-bit version of the driver.
 - a. For the 64-bit OLE DB driver, run:

```
wevtutil im "c:\Program Files\Vertica Systems\OLEDB64\lib\VerticaOLEDB64.man"  
/resourceFilePath:"c:\Program Files\Vertica Systems\OLEDB64\lib\vertica_8.1_oledb.dll"  
/messageFilePath:"c:\Program Files\Vertica Systems\OLEDB64\lib\vertica_8.1_oledb.dll"
```

- b. For the 32-bit OLE DB driver, run:

```
wevtutil im "c:\Program Files (x86)\Vertica Systems\OLEDB32\lib\VerticaOLEDB32.man"  
/resourceFilePath:"c:\Program Files (x86)\Vertica Systems\OLEDB32\lib\vertica_8.1_oledb.dll"  
/messageFilePath:"c:\Program Files (x86)\Vertica Systems\OLEDB32\lib\vertica_8.1_oledb.dll"
```

3. Run the command `wevtutil sl` to enable the logs.
 - a. For 64-bit OLE DB driver logs, run:

```
wevtutil sl VerticaOLEDB64/e:true
```

- b. For the 32-bit ODBC driver logs, run:

```
wevtutil sl VerticaOLEDB32/e:true
```

Note: Should you want to later disable the logs, you can use the same `wevtutil sl` command, substituting `/e:false` in place of `/e:true` when you issue the statement. Alternatively, you can enable or disable logs within the Windows Event Viewer itself.

Add the String Value LogType

By default, Vertica does not send OLE DB log entries to ETW. To activate ETW, add the string LogType to your Windows registry, and set its value to ETW.

1. Start the registry editor by typing `regedit.exe` in the Windows Run command box.
2. Navigate to the correct location in the registry.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\OLEDB\Driver
```

3. Right-click in the right pane of the **Registry Editor** window. Select **New** and then select **String Value**.
4. Change the name of the string value from `New Value #1` to `LogType`.
5. Double-click the new `LogType` entry. When prompted for a new value, enter `ETW`.
6. Exit the registry editor.

ETW is off by default. When ETW is activated, you can subsequently turn it off by clearing the value `ETW` from the `LogType` string.

Event Viewer Log Levels

The LogLevel parameter setting is described in the section, [Additional Parameter Settings](#). The parameter allows you to specify a LogLevel of 0 through 6. Be aware that Vertica compresses the log levels for the Windows Event Viewer. The six levels are compressed to four in Event Viewer.

Vertica LogLevel Setting	Vertica LogLevel Description	Entries are sent to Event Viewer as log level...	Event Viewer Displays...
0	(No logging)	0	(No logging)
1	Fatal Errors	1	Critical
2	Errors	2	Error
3	Warnings	3	Warning
4	Info	4	Information
5	Debug	4	
6	Trace (all messages)	4	

Examples:

- A LogLevel setting of 5 sends fatal errors, errors, warnings, info and debug log level entries to Event Viewer as Level 4 (Information).
- A LogLevel setting of 6 sends fatal errors, errors, warnings, debug and trace log level entries to Event Viewer as Level 4.

Where to Find Logs in Event Viewer

1. Launch the **Windows Event Viewer**.
2. From **Event Viewer (Local)**, expand **Applications and Services Logs**.
3. Expand the folder that contains the log you want to review (for example, `VerticaOLEDB64`).
4. Select the Vertica OLE DB log under the folder. Entries appear in the right pane.

Event Log Entry: Event ID

Once you have chosen an OLE DB log in Event Viewer, note the value in the **Event ID** field.

Each Event Log entry includes one of four Event IDs. An Event ID of 0 is informational (debug, info, and trace events), 1 is an error, 2 is a fatal event, and 3 is a warning.

The ADO.NET Driver for Windows

The Vertica ADO.NET driver for Windows is installed as part of the Client Drivers and Tools for Windows.

The ADO.NET driver is installed in the ADO.NET folder of the installation folder. The driver is also installed into the Windows Global Assembly Cache (GAC).

For information on how the ADO.NET driver integrates with Microsoft components previously installed on your system, see [Microsoft Components](#).

The Visual Studio Plug-in for Windows

The Visual Studio plug-in for Windows is installed as part of the Client Drivers and Tools for Windows.

For information on how the Visual Studio plug-in integrates with Microsoft components previously installed on your system, see [Microsoft Components](#).

Visual Studio Limitations

Visual Studio 2012 May Require Update 3

You may need to install update 3 to Visual Studio 2012 if:

- You launch Server Explorer to view and work with your Vertica server, but the Vertica data source is not visible.
- You create a SSAS cube, connect to Vertica, and find either an empty list of tables or tables not functioning correctly.

This issue does not occur for other versions of Visual Studio supported by Vertica.

Results Viewer Limited to 655 Columns

The Visual Studio results viewer cannot execute a query that includes more than 655 columns. If a table includes more than 655 columns, select specific columns (up to 655 total) rather than selecting all columns.

Manually Refresh Settings for Visual Studio

If, after installing the Visual Studio plug-in, you do not see Vertica listed as a data provider, manually refresh.

To do so, run `devenv.exe/setup`, which you can find in the Visual Studio installation folder.

SQL Pane Issues

- **ALTER TABLE or CREATE TABLE**

You use Visual Studio 2008, 2010, 2012, 2013, or 2015 and issue the ALTER TABLE or CREATE TABLE statement in the SQL pane. However, a message displays telling you that the statement is not supported. To resolve the error, click **Continue**, and the query executes.

- **Queries with Semicolons**

You use Visual Studio 2008, 2010, 2012, 2013, or 2015 and execute a SQL query in the SQL pane. If you include a semicolon (;) with your query, the query executes, but the result returned cannot be edited. To avoid this issue, enter the same query in the SQL pane without the semicolon.

- **Quoting Boolean Values**

You use Visual Studio 2008, 2010, 2012, 2013, or 2015 to connect to the Vertica database and execute a SQL query in the SQL pane. When attempting to insert a value into a Boolean column without putting quotes around the value, subsequent execution of the SQL statement returns an error. To work around this issue, include quotes.

Uninstalling Client Drivers and Tools for Windows Error

There is a scenario where an uninstall of the Client Drivers and Tools for Windows package fails with a message that the .NET framework is required. What follows is the scenario that causes this issue.

1. You Install the Client Drivers and Tools for Windows.
2. You then install Visual Studio 2010 or 2012, which includes installation of the .NET framework 4.0 or 4.5.
3. You uninstall the .NET framework using the Windows Control Panel.

4. You then attempt to uninstall the Client Drivers and Tools for Windows. The uninstall fails with the message that .NET framework is required.

Perform the following to correct this issue:

1. Reinstall the .NET framework 4.0 or 4.5 manually, using the Windows Control Panel.
2. Uninstall the Client Drivers and Tools for Windows.

Installing the Client Drivers on Mac OS X

This section details how to install the client drivers on Mac OS X.

Installing the JDBC Driver on Mac OS X

To install the Vertica JDBC driver on your Mac OS X client system, download the cross-platform JDBC driver `.jar` file to your system and ensure OS X's Java installation can find it.

Downloading the JDBC Driver

To download the Vertica JDBC driver on Mac OS X:

1. On your Mac client system, open a browser and log into the [myVertica portal](#).
2. Navigate to the Downloads page, scroll to the Client Software download section, and click the download link for the JDBC driver.
3. Accept the license agreement and wait for the download to complete.

Ensuring Java Can Find the JDBC Driver

In order for your Java client application to use the Vertica JDBC driver, the Java interpreter needs to be able to find its library file. Choose one of these methods to tell the Java interpreter where to look for the library:

- Copy the JDBC `.jar` file you downloaded to either the system-wide Java Extensions folder (`/Library/Java/Extensions`) or your user Java Extensions folder (`/Users/username/Library/Java/Extensions`).

- Add the directory containing the JDBC .jar file to the CLASSPATH environment variable (see [Modifying the Java CLASSPATH](#)).
- Specify the directory containing the JDBC .jar using the -cp argument in the Java command line you use to start your Java command line.

Installing the ODBC Driver on Mac OS X

You can obtain the Vertica ODBC driver for Mac OS X as a .pkg file through the [myVertica portal](#). You can run the installer as a regular [Mac OS X installer](#) or [silently](#). This driver is compatible with both 32-bit and 64-bit applications.

The installer is designed to be used with the standard iODBC Driver Manager included in Mac OS X. While Mac OS X ships with the iODBC Driver Manager already installed, you may choose to download the most recent version of the driver at the [iODBC.org](#) website.

By default, the installer installs the driver in the following location: `/Library/Vertica/ODBC/lib/libverticaodbc.dylib`. The installer also automatically registers a driver named "Vertica" with the iODBC Driver Manager.

To use the unixODBC Driver Manager instead of Apple's iODBC Driver Manager, see the [unixODBC.org](#) website.

Before You Download the Driver

If you installed a previous version of the Vertica ODBC driver for Mac OS X, your system may already have a registered driver named "Vertica." In this case, if you must remove or rename the older version of the driver before installing the Vertica ODBC driver .pkg.

To have multiple versions of the driver installed on your system at the same time, you must rename the currently installed version of the driver to something other than "Vertica." You can do so using the Apple [ODBC Administrator Tool](#).

To rename the driver:

1. Using your web browser, download and install the Apple ODBC Administrator Tool.
2. Locate and open the ODBC Administrator Tool after installation:
 - a. Navigate to **Finder > Applications > Utilities**.
 - b. Open the ODBC Administrator Tool.

3. Click the **Drivers** tab, and then select the driver named "Vertica."
4. Click the **Configure** button. A dialog box opens.
 - a. In Description, enter a new name for the driver, and then click **OK**. The dialog box closes.
 - b. On the ODBC Administrator page, click **Apply**.
5. Exit the ODBC Administrator Tool.

Download the Driver

Follow these steps to download the Vertica ODBC driver for Mac OS X:

1. On your Mac OS X client system, open a browser, and log in to the [myVertica portal](#).
2. Install the Vertica ODBC driver for Mac OS X:
 - a. Navigate to the Downloads tab, and scroll to the Client Software section.
 - b. Click the download link for the Mac OS X ODBC installer.
3. Accept the license agreement, and wait for the download to complete.

Install the Mac OS X ODBC Driver

As a Mac OS X Administrator, double-click the installer to start the installation. Follow the prompts as the wizard guides you through each step of the process.

Note: After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see [Creating an ODBC DSN for Macintosh OS X Clients](#).

Silently Install the Mac OS X ODBC Driver

1. Log into the client Mac in one of two ways:
 - As an administrator account, if you are installing the driver for system-wide use
 - As the user who needs to use the Vertica ODBC driver
2. Open a terminal window. In the Finder, click **Applications > Utilities > Terminal**.
3. Install the .pkg file containing the ODBC driver using the command:

```
sudo installer -pkg ~/Downloads/vertica-odbc7.0pkg -target /
```

In the preceding .pkg command, change the path to that of the downloaded file, *if*:

- You downloaded the driver .pkg file to a directory other than your Downloads directory.
- You downloaded the driver using another user account.

Note: After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see [Creating an ODBC DSN for Macintosh OS X Clients](#).

Uninstall the Mac OS X ODBC Driver

Uninstalling the Mac OS X ODBC Client-Driver does not remove any existing DSNs associated with the driver.

To uninstall:

1. Open a terminal window.
2. Enter the command:

```
sudo /Library/Vertica/ODBC/bin/Uninstall
```

Upgrade or Downgrade the Mac OS X ODBC Driver

All installations of the Vertica ODBC driver for Mac OS X are uniquely identified by a package ID and version number. The package ID does not change between versions, but the version number does. If you attempt multiple installations of the same version of the driver, a name collision error occurs. Therefore, multiple installations of the same version of the driver cannot coexist on a single operating system.

- **Upgrading**—Newly installed versions of the Vertica ODBC driver for Mac OS X automatically upgrade the relevant driver system settings. Any DSNs associated with a previous version of the driver are not affected, except that they begin using the newer version of the driver.
- **Downgrading**—Run the uninstall script to remove the current version of the Vertica ODBC driver for Mac OS X. Complete this step before installing an older driver version.

ODBC Driver Settings on Mac OS X

ODBC driver settings are automatically configured using the Vertica ODBC driver installer on Mac OS X. It is not necessary to configure additional ODBC driver settings on Mac OS X platforms beyond what is automatically configured by the installer. You can, however, set the ODBC driver settings by editing the VERTICAINI environment variable in each user's ~/.MacOSX/environment.plist file. See the [Environment Variables](#) entry in the Apple Developer's Library for more information.

See [Additional Parameter Settings](#) for a list of the additional settings.

Creating an ODBC Data Source Name (DSN)

A Data Source Name (DSN) is the logical name that is used by Open Database Connectivity (ODBC) to refer to the driver and other information that is required to access data from a data source. Whether you are developing your own ODBC client code or you are using a third-party tool that needs to access Vertica using ODBC, you need to configure and test a DSN. The method you use depends upon the client operating system you are using.

Refer to the following sections for information specific to your client operating system.

- [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)
- [Creating an ODBC DSN for Windows Clients](#)
- [Creating an ODBC DSN for Macintosh OS X Clients](#)
- [Data Source Name \(DSN\) Connection Properties](#)
- [Setting DSN Connection Properties](#)

Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX

You define DSN on Linux, Solaris, and other UNIX-like platforms in a text file. Your client's driver manager reads this file to determine how to connect to your Vertica database. The driver manager usually looks for the DSN definitions in two places:

- `/etc/odbc.ini`
- `~/odbc.ini` (a file named `.odbc.ini` in the user's home directory)

Users must be able to read the `odbc.ini` file in order to use it to connect to the database. If you use a global `odbc.ini` file, consider creating a UNIX group with read access to the file. Then add the users who need to use the DSN to this group.

The structure of these files is the same, only their location differs. If both files are present, the `~/odbc.ini` file usually overrides the system-wide `/etc/odbc.ini` file.

Note: See your ODBC driver manager's documentation for details on where these files should be located and any other requirements.

odbc.ini File Structure

The `odbc.ini` is a text file that contains two types of lines:

- Section definitions, which are text strings enclosed in square brackets.
- Parameter definitions, which contain a parameter name, an equals sign (=), and then the parameter's value.

The first section of the file is always named `[ODBC Data Sources]`, and contains a list of all the DSNs that the `odbc.ini` file defines. The parameters in this section are the names of the DSNs, which appear as section definitions later in the file. The value is a text description of the DSN and has no function. For example, an `odbc.ini` file that defines a single DSN named `VerticaDSN` could have this ODBC Data Sources section:

```
[ODBC Data Sources]
HPVerticaDSN = "vmartdb"
```

Appearing after the ODBC data sources section are sections that define each DSN. The name of a DSN section must match one of the names defined in the ODBC Data Sources section.

Configuring the odbc.ini file:

To create or edit the DSN definition file:

1. Using the text editor of your choice, open `odbc.ini` or `~/odbc.ini`.
2. Create an ODBC Data Sources section and define a parameter:
 - Whose name is the name of the DSN you want to create
 - Whose value is a description of the DSN

For example, to create a DSN named `VMart`, you would enter:

```
[ODBC Data Sources]
VMart = "VMart database on Vertica"
```

3. Create a section whose name matches the DSN name you defined in step 2. In this section, you add parameters that define the DSN's settings. The most commonly-defined parameters are:

- **Description** – Additional information about the data source.
- **Driver** – The location and designation of the Vertica ODBC driver, or the name of a driver defined in the `odbcinst.ini` file (see below). For future compatibility, use the name of the symbolic link in the library directory, rather than the library file:
 - `/opt/vertica/lib`, on 32-bit clients
 - `/opt/vertica/lib64`, on 64-bit clients

For example, the symbolic link for the 64-bit ODBC driver library is:

```
/opt/vertica/lib64/libverticaodbc.so
```

The symbolic link always points to the most up-to-date version of the Vertica client ODBC library. Use this link so that you do not need to update all of your DSNs when you update your client drivers.

- **Database** – The name of the database running on the server. This example uses `vmartdb` for the `vmartdb`.
- **ServerName** — The name of the server where Vertica is installed. Use `localhost` if Vertica is installed on the same machine.

You can provide an IPv4 address, IPv6 address, or host name.

In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the `PreferredAddressFamily` option to force the connection to use either IPv4 or IPv6.

- **UID** — Either the database superuser (same name as database administrator account) or a user that the superuser has created and granted privileges. This example uses the user name `dbadmin`.
- **PWD** — The password for the specified user name. This example leaves the password field blank.
- **Port** — The port number on which Vertica listens for ODBC connections. For example, `5433`.
- **ConnSettings** — Can contain SQL commands separated by a semicolon. These commands can be run immediately after connecting to the server.

- **SSLKeyFile** — The file path and name of the client's private key. This file can reside anywhere on the system.
- **SSLCertFile** —The file path and name of the client's public certificate. This file can reside anywhere on the system.
- **Locale** — The default locale used for the session. By default, the locale for the database is: en_US@collation=binary (English as in the United States of America). Specify the locale as an ICU Locale. See the ICU User Guide (<http://userguide.icu-project.org/locale>) for a complete list of parameters that can be used to specify a locale.
- **PreferredAddressFamily:**

The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:

- ipv4—Connect to the server using IPv4.
- ipv6—Connect to the server using IPv6.
- none—Use the IP address provided by the DNS server.

For example:

```
[VMart]
Description = Vmart Database
Driver = /opt/vertica/lib64/libverticaodbc.so
Database = vmartdb
Servername = host01
UID = dbadmin
PWD =
Port = 5433
ConnSettings =
AutoCommit = 0
SSLKeyFile = /home/dbadmin/client.key
SSLCertFile = /home/dbadmin/client.crt
Locale = en_US@collation=binary
```

See [Data Source Name \(DSN\) Connection Properties](#) for a complete list of parameters including Vertica-specific ones.

Using an odbcinst.ini File

Instead of giving the path of the ODBC driver library in your DSN definitions, you can use the name of a driver defined in the `odbcinst.ini` file. This method is useful method if you have many DSNs and often need to update them to point to new driver libraries. It also allows you to set some additional ODBC parameters, such as the threading model.

Just as in the `odbc.ini` file, `odbcinst.ini` has sections. Each section defines an ODBC driver that can be referenced in the `odbc.ini` files.

In a section, you can define the following parameters:

- **Description** — Additional information about the data source.
- **Driver** — The location and designation of the Vertica ODBC driver, such as `/opt/vertica/lib64/libverticaodbc.so`

For example:

```
[HPVertica]
Description = Vertica ODBC Driver
Driver = /opt/vertica/lib64/libverticaodbc.so
```

Then, in your `odbc.ini` file, use the name of the section you created in the `odbcinst.ini` file that describes the driver you want to use. For example:

```
[VMart]
Description = Vertica Vmart database
Driver = HPVertica
```

If you are using the `unixODBC` driver manager, you should also add an ODBC section to override its standard threading settings. By default, `unixODBC` serializes all SQL calls through ODBC, which prevents multiple parallel loads. To change this default behavior, add the following to your `odbcinst.ini` file:

```
[ODBC]
Threading = 1
```

Configuring Additional ODBC Settings

On Linux and UNIX systems, you need to configure some additional driver settings before you can use your DSN. See [Required ODBC Driver Configuration Settings for Linux and UNIX](#) for details.

Testing an ODBC DSN Using `Isql`

The `unixODBC` driver manager includes a utility named `isql`, which is a simple ODBC command-line client. It lets you to connect to a DSN to send commands and receive results, similarly to `vsq`.

To use `isql` to test a DSN connection:

1. Run the following command:

```
$ isql -v DSNname
```

Where *DSNname* is the name of the DSN you created.

A connection message and a SQL prompt display. If they do not, you could have a configuration problem or you could be using the wrong user name or password.

2. Try a simple SQL statement. For example:

```
SQL> SELECT table_name FROM tables;
```

The isql tool returns the results of your SQL statement.

Note: If you have not set the `ErrorMessagesPath` in the additional driver configuration settings, any errors during testing will trigger a missing error message file ("The error message `NoSQLGetPrivateProfileString` could not be found in the en-US locale"). See [Required ODBC Driver Configuration Settings for Linux and UNIX](#) for more information.

Creating an ODBC DSN for Windows Clients

To create a DSN for Microsoft Windows clients, you must perform the following tasks:

- [Setting Up an ODBC DSN](#)
- [Testing the DSN Using Excel](#)

Setting Up an ODBC DSN

A *Data Source Name (DSN)* is the ODBC logical name for the drive and other information the database needs to access data. The name is used by Internet Information Services (IIS) for a connection to an ODBC data source.

This section describes how to use the Vertica ODBC Driver to set up an ODBC DSN. This topic assumes that the driver is already installed, as described in [Installing Client Drivers on Windows](#).

To set up a DSN

1. Open the ODBC Administrator. For example, you could navigate to **Start > Control Panel > Administrative Tools > Data Sources (ODBC)**.

Note: The method you use to open the ODBC Administrator depends on your version of Windows. Differences between Windows versions and **Start Menu** customizations could require you to take a different action to open the ODBC Administrator.

2. Decide if you want all users on your client system to be able to access to the DSN for the Vertica database.
 - If you want all users to have access, then click the **System DSN** tab.
 - Otherwise, click the **User DSN** tab to create a DSN that is only usable by your Windows user account.
3. Click **Add** to create a new DSN to connect to the Vertica database.
4. Scroll through the list of drivers in the Create a New Data Source dialog box to locate the Vertica driver. Select the driver, and then click **Finish**.

Note: If you have installed more than one version of the Vertica client drivers on your Windows client system, you may see multiple versions of the driver in this list. Choose the version that you know is compatible with your client application and Vertica Analytics Platform server. If you are unsure, use the latest version of the driver.

The Vertica ODBC DSN configuration dialog box appears.

5. Click the **More >>>** button to view a description of the field you are editing and the connection string defined by the DSN.
6. Enter the information for your DSN. The following fields are required:
 - **DSN Name** — The name for the DSN. Clients use this name to identify the DSN to which they want to connect. The DSN name must satisfy the following requirements:
 - Its maximum length is 32 characters.
 - It is composed of ASCII characters except for the following: [] { } , ; ? * = ! @ \
 - It contains no spaces.

- **Server** — The host name or IP address of the Vertica server to which you want to connect. Use localhost, if Vertica is installed on the same machine.

You can provide an IPv4 address, IPv6 address, or host name.

In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.

The PreferredAddressFamily option is available on the Client Settings tab.

- **Backup Servers** — A comma-separated list of host names or IP addresses used to connect to if the server specified by the Server field is down. Optional.
- **Database** — The name of the Vertica database.
- **User Name** — The name of the user account to use when connecting to the database. If the application does not supply its own user name when connecting to the DSN, this account name is used to log into the database.

The rest of the fields are optional. See [DSN Parameters](#) for detailed information about the DSN parameters you can define.

7. If you want to test your connection:
 - a. Enter at least a valid **DSN name**, **Server name**, **Database**, and either **User name** or select **Windows authentication**.
 - b. If you have not selected **Windows authentication**, you can enter a password in the **Password** box. Alternately, you can select **Password prompt** to have the driver prompt you for a password when connecting.
 - c. Click **Test Connection**.
8. When you have finished editing and testing the DSN, click **OK**. The Vertica ODBC DSN configuration window closes, and your new DSN is listed in the ODBC Data Source Administrator window.
9. Click **OK** to close the ODBC Data Source Administrator.

After creating the DSN, you can test it using [Microsoft Excel 2007](#).

Setting up a 32-Bit DSN on 64-Bit Versions of Microsoft Windows

On 64-bit versions of Windows, the default ODBC Data Source Administrator creates and edits DSNs that are associated with the 64-bit Vertica ODBC library.

Attempting to use these 64-bit DSNs with a 32-bit client application results in an architecture mismatch error. Instead, you must create a specific 32-bit DSN for 32-bit clients by running the 32-bit ODBC Administrator usually located at:

```
c:\Windows\SysWOW64\odbcad32.exe
```

This administrator window edits a set of DSNs that are associated with the 32-bit ODBC library. You can then use your 32-bit client applications with the DSNs you create with this version of the ODBC administrator.

Encrypting Passwords on ODBC DSN

When you install an ODBC driver and create a Data Source Name (DSN) the DSN settings are stored in the registry, including the password. Encrypting passwords on ODBC DSN applies only to Windows systems.

Encrypting passwords on an ODBC data source name (DSN) provides security against unauthorized database access. The password is not encrypted by default and is stored in plain-text.

Note: Password encryption applies only to new or modified ODBC DSNs. If you have a DSN created in Version 8.0 or earlier and upgrade to 8.1, the password does not get encrypted regardless of the encryption settings.

Enable Password Encryption

Use the `EncryptPassword` parameter to enable or disable password encryption for an ODBC DSN:

- `EncryptPassword = true` enables password encryption
- `EncryptPassword = false` (default) disables password encryption

Set `EncryptPassword` in the Windows registry - `HKEY_LOCAL_MACHINE > Software > Vertica > ODBC > Driver EncryptPassword=<true/false>`.

Note: For 32 bit driver running on 64 bit windows verify password encryption here:

```
HKEY_LOCAL_MACHINE > Software > Wow6432Node > Vertica > ODBC >  
Driver > EncryptPassword=<true/false>
```

Encrypted passwords get updated in the following registry locations:

For a user DSN:

```
HKEY_CURRENT_USER-> Software -> ODBC -> ODBC.INI -> DSNNAME -> PWD
```

For a system DSN:

```
HKEY_LOCAL_MACHINE-> Software -> ODBC -> ODBC.INI -> DSNNAME -> PWD
```

Verify Password Encryption

Use Windows Registry editor to determine if password encryption is enabled based on the value of `EncryptPassword`. Depending on the type of DSN you installed, check the following:

For a user DSN: `HKEY_CURRENT_USER > Software > ODBC > ODBC.INI > dsn name > isPasswordEncrypted=<1/0>`

For a system DSN: `HKEY_LOCAL_MACHINE > Software > ODBC > ODBC.INI > dsn name > isPasswordEncrypted=<1/0>`

For each DSN, the value of the `isPasswordEncrypted` parameter indicates the status of the password encryption, where 1 indicates an encrypted password and 0 indicates an unencrypted password.

Testing an ODBC DSN Using Excel

You can use Microsoft Excel to verify that an application can connect to an ODBC data source or other ODBC application.

1. Open Microsoft Excel, and select **Data > Get External Data > From Other Sources > From Microsoft Query**.
2. When the Choose Data Source dialog box opens:
 - a. Select **New Data Source**, and click **OK**.
 - b. Enter the name of the data source.

- c. Select the Vertica driver.
- d. Click **Connect**.
3. When the Vertica Connection Dialog box opens, enter the connection information for the DSN, and click **OK**.
4. Click **OK** on the Create New Data Source dialog box to return to the Choose Data Source dialog box.
5. Select VMart_Schema*, and verify that the Use the Query Wizard check box is deselected. Click **OK**.
6. When the Add Tables dialog box opens, click **Close**.
7. When the Microsoft Query window opens, click the **SQL** button.
8. In the SQL window, write any simple query to test your connection. For example:

```
SELECT DISTINCT calendar_year FROM date_dimension;
```

9.
 - If you see the caution, "SQL Query can't be represented graphically. Continue anyway?" click **OK**.
 - The data values 2003, 2004, 2005, 2006, 2007 indicate that you successfully connected to and ran a query through ODBC.
10. Select **File > Return Data to Microsoft Office Excel**.
11. In the Import Data dialog box, click **OK**.

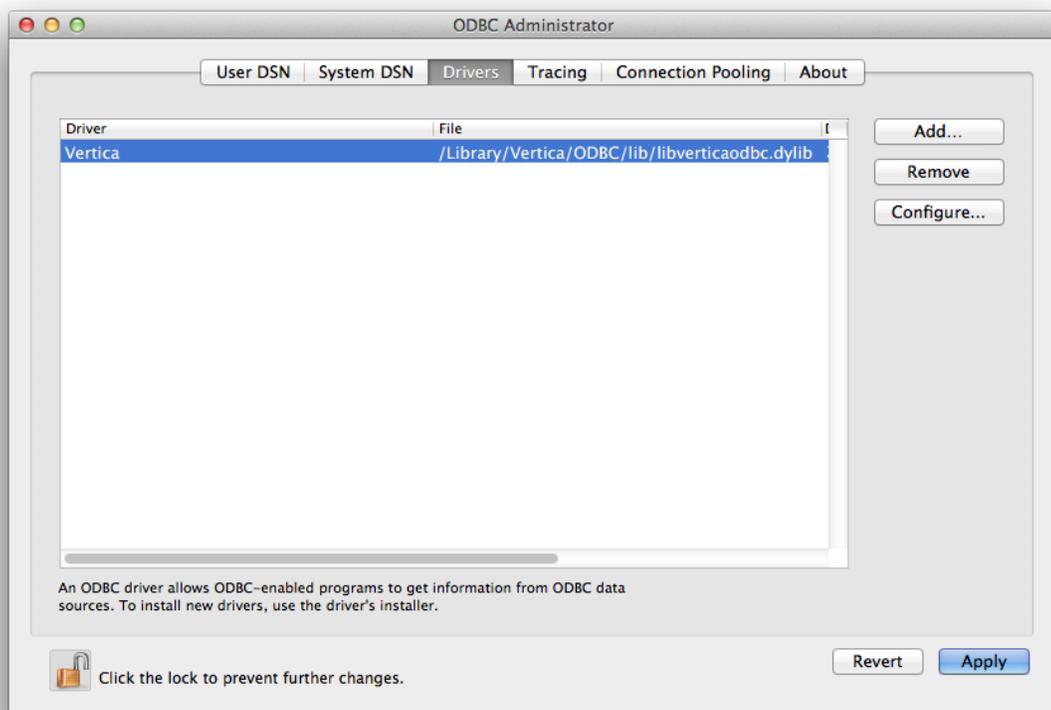
The data is now available for use in an Excel worksheet.

Creating an ODBC DSN for Macintosh OS X Clients

You can use the Vertica ODBC Driver to set up an ODBC DSN. This procedure assumes that the driver is already installed, as described in [Installing the ODBC Driver on Macintosh OS X](#).

Setting Up a DSN

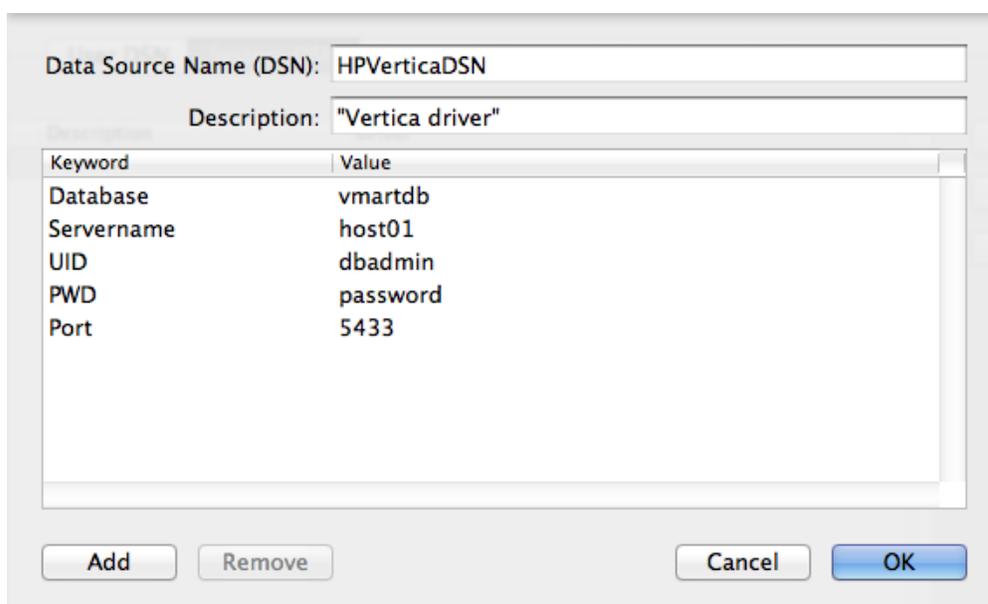
1. Using your web browser, download and install the Apple [ODBC Administrator Tool](#).
2. Locate and open the ODBC Administrator Tool after installation:
 - a. Navigate to **Finder > Applications > Utilities**.
 - b. Open the ODBC Administrator Tool.
3. Click the **Drivers** tab, and verify that the Vertica driver is installed.



4. Specify if you want all users on your client system to be able to access the DSN for the Vertica database:
 - If you want all users to have access, then click the **System DSN** tab.
 - Otherwise, click the **User DSN** tab to create a DSN that is only usable by your Macintosh user account.

5. Click **Add...** to create a new DSN to connect to the Vertica database.
6. Scroll through the list of drivers in the Choose A Driver dialog box to locate the Vertica driver. Select the driver, and then click **OK**. A dialog box opens that requests DSN parameter information.
7. In the dialog box, enter the **Data Source Name (DSN)** and an optional **Description**. To do so, click **Add** to insert keywords (parameters) and values that define the settings needed to connect to your database. Then, click **OK**.

The following figure shows the minimum parameters that are required to configure a DSN so that it connects to the database:



The screenshot shows the 'Data Source Name (DSN)' dialog box. The 'Data Source Name (DSN)' field contains 'HPVerticaDSN' and the 'Description' field contains '"Vertica driver"'. Below these fields is a table with two columns: 'Keyword' and 'Value'. The table contains the following entries:

Keyword	Value
Database	vmartdb
Servename	host01
UID	dbadmin
PWD	password
Port	5433

At the bottom of the dialog box, there are four buttons: 'Add', 'Remove', 'Cancel', and 'OK'.

8. In the ODBC Administrator dialog box, click **Apply**.

See [Data Source Name \(DSN\) Connection Properties](#) for a complete list of parameters including those specific to Vertica.

After configuring the ODBC Administrator Tool, you may need to configure additional driver settings before you can use your DSN, depending on your environment. See [Additional ODBC Driver Configuration Settings](#) for details.

Note: If you want to test your connection, use the `iodbctest` utility. For the procedure, see [Testing a DSN Using `iodbctest`](#).

Testing an ODBC DSN Using iodbctest

The standard iODBC Driver Manager on OS X includes a utility named `iodbctest` that lets you test a DSN to verify that it is correctly configured. You pass this command a connection string in the same format that you would use to open an ODBC database connection. After configuring your DSN connection, you can run a query to verify that the connection works.

For example:

```
# iodbctest "DSN=VerticaDSN;UID=dbadmin;PWD=password"
iODBC Demonstration program
This program shows an interactive SQL processor
Driver Manager: 03.52.0607.1008
Driver: 07.01.0200 (verticaodbcw.so)
SQL> SELECT table_name FROM tables;
table_name
-----
customer_dimension
product_dimension
promotion_dimension
date_dimension
vendor_dimension
employee_dimension
shipping_dimension
warehouse_dimension
inventory_fact
store_dimension
store_sales_fact
store_orders_fact
online_page_dimension
call_center_dimension
online_sales_fact
numbers
result set 1 returned 16 rows.
```

Data Source Name (DSN) Connection Properties

The following tables list the connection properties you can set in the DSNs for use with Vertica's ODBC driver.

Required Connection Properties

These connection properties are the minimum required to create a functioning DSN.

NOTE: If you use a host name (Servername) whose DNS entry resolves to multiple IP addresses, the client attempts to connect to the first IP address returned by the DNS. If a connection

cannot be made to the first address, the client attempts to connect to the second, then the third, continuing until it either connects successfully or runs out of addresses.

Property	Description	Default Value
Driver	The file path and name of the driver used.	none
Database	The name of the database running on the server.	none
Servename	<p>The host name or IP address of any active node in a Vertica cluster.</p> <p>You can provide an IPv4 address, IPv6 address, or host name.</p> <p>In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.</p> <p>You can also use the aliases "server" and "host" for this property.</p>	none
UID	The database username.	none

Optional Properties

These are basic properties that are optional.

Property	Description	Default Value
Port	The port number on which Vertica listens for ODBC connections.	5433
PWD	The password for the specified user name. You may insert an empty string to leave this property blank.	none (login only succeeds if the user does not have a password set)

Property	Description	Default Value
PreferredAddressFamily	<p>The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:</p> <ul style="list-style-type: none"> • ipv4—Connect to the server using IPv4. • ipv6—Connect to the server using IPv6. • none—Use the IP address provided by the DNS server. 	none
Protocol	<p>Specifies the front-end protocol that the ODBC client driver uses to communicate with a previous Vertica server version. For the 8.1.x ODBC client driver to connect to Vertica server version 7.1, specify <code>Protocol=3.5</code>. Specifying the protocol, while not required, allows full backwards compatibility.</p> <p>Note: The default protocol version, 3.6, applies to Vertica server versions 7.2, 8.0, and 8.1.x. You do not need to specify the protocol property when connecting from an 8.1.x ODBC client driver to server versions 7.2, 8.0, and 8.1.x.</p>	3.6

Advanced Settings

Property	Description	Default
AutoCommit	A Boolean value that controls whether the driver automatically commits transactions after executing a DML statement.	true
BackupServerNode	A string containing the	none

Property	Description	Default
	<p>host name or IP address that client libraries can try to connect to if the host specified in ServerName is unreachable. Connection attempts continue until successful or until the list of server nodes is exhausted.</p> <p>Valid values: Comma-separated list of servers optionally followed by a colon and port number.</p>	
ConnectionLoadBalance	<p>A Boolean value that indicates whether the connection can be redirected to a host in the database other than the ServerNode.</p> <p>This affects the connection only if the load balancing is set to something other than "none". When the node differs from the node the client is connected to, the client disconnects and reconnects to the targeted node. See About Native Connection Load Balancing in the Administration Guide.</p>	false
ConnSettings	A string containing SQL commands that the driver should execute	none

Property	Description	Default
	<p>immediately after connecting to the server. You can use this property to configure the connection, such as setting a schema search path.</p> <p>Reserved symbol: In the connection string ';' is a reserved symbol. To set multiple properties as part of ConnSettings properties, use '%3B' for ';'. Also use '+' for spaces.</p>	
ConvertSquareBracketIdentifiers	Controls whether square-bracket query identifiers are converted to a double quote identifier for compatibility when making queries to a Vertica database.	false
DirectBatchInsert	<p>A Boolean value that controls where data inserted through the connection is stored. When set to true, Vertica directly inserts data into ROS containers. Otherwise, it stores data using AUTO mode.</p> <p>When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details</p>	false

Property	Description	Default
	<p>about load options, see Choosing a Load Method.</p>	
DriverStringConversions	<p>Controls whether the ODBC driver performs type conversions on strings sent between the ODBC driver and the database. Possible values are:</p> <ul style="list-style-type: none"> • NONE—No conversion in either direction. This results in the highest performance. • INPUT—Strings sent from the client to the server are converted, but strings sent from the server to the client are not. • OUTPUT—Strings sent by the server to the client are converted, but strings sent from the client to the server are not. • BOTH—Strings are converted in both directions. 	OUTPUT
Locale	<p>The locale used for the session. Specify the locale as an ICU Locale.</p> <p>See: The ICU User Guide (http://userguide.icu-project.org/locale) for a</p>	en_US@collation=binary (English as in the United States of America)

Property	Description	Default
	complete list of properties that can be used to specify a locale.	
PromptOnNoPassword	[Windows only] Controls whether users are prompted to enter a password, if none is supplied by the connection string or DSN used to connect to Vertica. See Prompting Windows Users for Passwords .	false
ReadOnly	A true or false value that controls whether the connection can read data only from Vertica.	false
ResultBufferSize	Size of memory buffer for the large result sets in streaming mode. A value of 0 means ResultBufferSize is turned off. This property was previously called MaxMemoryCache	131072 (128KB)
TransactionIsolation	Sets the transaction isolation for the connection. Valid values are: <ul style="list-style-type: none"> • Read Committed • Serializable • Server Default 	Server Default

Property	Description	Default
	See Changing Transaction Isolation Levels in the Administrator's Guide for an explanation of transaction isolation.	

Identification

Property	Description	Default	Standard/ Vertica
Description	Description for the DSN entry. Required? No Insert an empty string to leave the description empty.	none	Standard
Label / SessionLabel	Sets a label for the connection on the server. This value appears in the session_id column of the V_MONITOR.SESSIONS system table. Label and SessionLabel are synonyms and can be used interchangeably.	none	Vertica

Encryption

Property	Description	Default	Standard/ Vertica
SSLMode	Controls whether the connection to the database uses SSL encryption. Valid values follow. For descriptions of these values, refer to Configuring SSL for ODBC Clients . <ul style="list-style-type: none"> require prefer—Prefers that the server use SSL. If the server does not offer an encrypted channel, 	prefer	Vertica

Property	Description	Default	Standard/ Vertica
	<p>the client requests one. The first connection attempt to the database tries to use SSL. If that attempt fails, a second connection is attempted over a clear channel.</p> <ul style="list-style-type: none"> • allow—Makes a connection to the server whether the server uses SSL or not. The first connection attempt to the database is attempted over a clear channel. If that fails, a second connection is attempted over SSL. • disable—Never connects to the server using SSL. Typically, you use this setting for troubleshooting. 		
SSLCertFile	The absolute path of the client's public certificate file. This file can reside anywhere on the system.	none	Vertica
SSLKeyFile	The absolute path to the client's private key file. This file can reside anywhere on the system.	none	Vertica

Third-Party Compatibility

Property	Description	Default	Standard/ Vertica
ColumnsAsChar	Specifies how character column types are reported when the driver is in Unicode mode. When set to false, the ODBC driver reports the data type of character columns as WCHAR. If you set ColumnsAsChar to true, the driver identifies character column as	false	Vertica

Property	Description	Default	Standard/ Vertica
	<p>CHAR.</p> <p>You typically use this setting for compatibility with some third-party clients, such as Informatica.</p>		
ThreePartNaming	<p>A Boolean value that controls how catalog names are interpreted by the driver. When this value is false, the driver reports that catalog names are not supported. When catalog names are not supported, they cannot be used as a filter in database metadata API calls. In this case, the driver returns NULL as the catalog name in all driver metadata results.</p> <p>When this value is true, catalog names can be used as a filter in database metadata API calls. In this case, the driver returns the database name as the catalog name in metadata results. Some third-party applications assume a certain catalog behavior and do not work properly with the default values. Enable</p>	<p>false (UNIX)</p> <p>true (Window)</p>	Vertica

Property	Description	Default	Standard/ Vertica
	<p>this option if your client software expects to get the catalog name from the database metadata and use it as part of a three-part name reference.</p>		
<p>EnforceBatchInsertNullConstraints</p>	<p>Prevents NULL values from being loaded into columns with a NOT NULL constraint during batch inserts. When this value is set to true, batch inserts roll back when NULL values are inserted in to columns with NOT NULL constraints. When this value is set to false, batch insert behavior is unchanged.</p> <p>Micro Focus International plc recommends only using this property with SAP Data Services as it could negatively impact database performance.</p>	<p>false</p>	<p>Vertica</p>

Kerberos Connection Properties

Use the following properties for client authentication using Kerberos.

Property	Description	Default	Standard/ Vertica
KerberosServiceName	Provides the service name portion of the Vertica Kerberos principal; for example: <code>vertica/host@EXAMPLE.COM</code>	vertica	Vertica
KerberosHostname	Provides the instance or host name portion of the Vertica Kerberos principal; for example: <code>vertica/host@EXAMPLE.COM</code>	Value specified in the servername connection string property	Vertica

See Also

- [Required ODBC Driver Configuration Settings for Linux and UNIX](#)

Setting DSN Connection Properties

The properties in the following tables are common for all user and system DSN entries. The examples provided are for Windows clients.

To edit DSN properties:

- On UNIX and Linux client platforms, you can edit the `odbc.ini` file. The location of this file is specific to the driver manager. See [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#).
- On Windows client platforms, you can edit some DSN properties using the Vertica ODBC client driver interface. See [Creating an ODBC DSN for Windows Clients](#).
- You can also edit the DSN properties directly by opening the DSN entry in the Windows registry (for example, at `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\DSNname`). Directly editing the registry can be risky, so you should only use this method for properties that cannot be set through the ODBC driver's user interface, or via your client code.
- You can set properties in the connection string when opening a connection using the `SQLDriverConnect()` function:

```
sqlRet = SQLDriverConnect(sql_hDBC, 0, (SQLCHAR*)"DSN=DSNName;Locale=en_GB@collation=binary", SQL_
NTS, szDNS, 1024,&nSize, SQL_DRIVER_NOPROMPT);
```

Note: In the connection string ';' is a reserved symbol. If you need to set multiple properties as part of the ConnSettings property use '%3B' in place of ';'. Also use '+' instead of spaces.

For example:

```
sqlRet = SQLDriverConnect(sql_hDBC, 0, (SQLCHAR*)"DSN=VerticaSQL;ConnSettings=set+search_
path+to+a,b,c%3Bset+locale=ch;SSLMode=prefer", SQL_NTS,
szDNS, 1024,&nSize, SQL_DRIVER_NOPROMPT);
```

- Your client code can retrieve DSN property values after a connection has been made to Vertica using the SQLGetConnectAttr() and SQLGetStmtAttr() API calls. Some properties can be set and using SQLSetConnectAttr() and SQLSetStmtAttr().

For details of the list of properties specific to Vertica see [ODBC Header Files specific to Vertica](#).

Programming ODBC Client Applications

Vertica provides an Open Database Connectivity (ODBC) driver that allows applications to connect to the Vertica database. This driver can be used by custom-written client applications that use the ODBC API to interact with Vertica. ODBC is also used by many third-party applications to connect to Vertica, including business intelligence applications and extract, transform, and load (ETL) applications.

This section details the process for configuring the Vertica ODBC driver. It also explains how to use the ODBC API to connect to Vertica in your own client applications.

This section assumes that you have already installed the ODBC libraries on your client system. If you have not, see [Client Drivers](#).

ODBC Architecture

The ODBC architecture has four layers:

- **Client Application**

Is an application that opens a data source through a Data Source Name (DSN). It then sends requests to the data source, and receives the results of those requests. Requests are made in the form of calls to ODBC functions.

- **Driver Manager**

Is a library on the client system that acts as an intermediary between a client application and one or more drivers. The driver manager:

- Resolves the DSN provided by the client application.
- Loads the driver required to access the specific database defined within the DSN.
- Processes ODBC function calls from the client or passing them to the driver.
- Retrieves results from the driver.
- Unloads drivers when they are no longer needed.

On Windows and Mac client systems, the driver manager is provided by the operating system. On Linux and UNIX systems, you usually need to install a driver manager. See [ODBC](#)

[Prerequisites](#) for a list of driver managers that can be used with Vertica on your client platform.

- **Driver**

A library on the client system that provides access to a specific database. It translates requests into the format expected by the database, and translates results back into the format required by the client application.

- **Database**

The database processes requests initiated at the client application and returns results.

ODBC Feature Support

The ODBC driver for Vertica supports the most of the features defined in the Microsoft ODBC 3.5 specifications. The following features are *not* supported:

- Updatable result sets
- Backwards scrolling cursors
- Cursor attributes
- More than one open statement per connection. For example you cannot execute a new statement while another statement has a result set open. If you need to execute multiple statements at once, open multiple database connections.
- Keysets
- Bookmarks

The Vertica ODBC driver accurately reports its capabilities. If you need to determine whether it complies with a specific feature, you should query the driver's capabilities directly using the `SQLGetInfo()` function.

ODBC Header File Specific to Vertica

The Vertica ODBC driver provides a C header file named `verticaodbc.h` that defines several useful constants that you can use in your applications. These constants let you access and alter settings specific to Vertica.

This file's location depends on your client operating system:

- /opt/vertica/include on Linux and UNIX systems.
- C:\Program Files (x86)\Vertica\ODBC\include on Windows systems.

The constants defined in this file are listed below.

Parameter	Description	Associated Function
SQL_ATTR_VERTICA_RESULT_BUFFER_SIZE	Sets the size of the buffer used when retrieving results from the server.	SQLSetConnectAttr() SQLGetConnectAttr()
SQL_ATTR_VERTICA_DIRECT_BATCH_INSERT	<p>Determines whether a batch is inserted directly into the ROS (1) or using AUTO mode (0). By default batches are inserted using AUTO.</p> <p>When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</p>	SQLSetConnectAttr() SQLSetStmtAttr() SQLGetConnectAttr() SQLGetStmtAttr()
SQL_ATTR_VERTICA_LOCALE	Changes the locale from en_US@collation=binary to the ICU locale specified. See Setting the Locale for ODBC Sessions for an example of using this parameter.	SQLSetConnectAttr() SQLGetConnectAttr()

Connecting to the Database

The first step in any ODBC application is to connect to the database. When you create the connection to a data source using ODBC, you use the name of the DSN that contains the details of the driver to use, the database host, and other basic information about connecting to the data source.

There are 4 steps your application needs to take to connect to a database:

1. Call `SQLAllocHandle()` to allocate a handle for the ODBC environment. This handle is used to create connection objects and to set application-wide settings.
2. Use the environment handle to set the version of ODBC that your application wants to use. This ensures that the data source knows which API your application will use to interact with it.
3. Allocate a database connection handle by calling `SQLAllocHandle()`. This handle represents a connection to a specific data source.
4. Use the `SQLConnect()` or `SQLDriverConnect()` functions to open the connection to the database.

Note: If you specify a locale either in the connection string or in the DSN, the call to the connection function returns `SQL_SUCCESS_WITH_INFO` on a successful connection, with messages about the state of the locale.

When creating the connection to the database, use `SQLConnect()` when the only options you need to set at connection time is the username and password. Use `SQLDriverConnect()` when you want to change connection options, such as the locale.

The following example demonstrates connecting to a database using a DSN named `ExampleDB`. After it creates the connection successfully, this example simply closes it.

```
// Demonstrate connecting to Vertica using ODBC.
// Standard i/o library
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// SQL include files that define data types and ODBC API
// functions
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
int main()
```

```
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hdlEnv; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }

    // Set the ODBC version we are going to use to
    // 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_INTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC 3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application version to ODBC 3.\n");
    }

    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated Database handle.\n");
    }

    // Connect to the database using
    // SQL Connect
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char *userID = "ExampleUser";
    const char *passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
        SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
        (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not connect to database.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Connected to database.\n");
    }

    // We're connected. You can do real
    // work here

    // When done, free all of the handles to close them
    // in an orderly fashion.
    printf("Disconnecting and freeing handles.\n");
    ret = SQLDisconnect(hdlDbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Error disconnecting from database. Transaction still open?\n");
        exit(EXIT_FAILURE);
    }

    SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
}
```

```
    exit(EXIT_SUCCESS);  
}
```

Running the above code prints the following:

```
Allocated an environment handle.  
Set application version to ODBC 3.  
Allocated Database handle.  
Connecting to database.  
Connected to database.  
Disconnecting and freeing handles.
```

See [Setting the Locale for ODBC Sessions](#) for an example of using `SQLDriverConnect` to connect to the database.

Notes

- If you use the DataDirect[®] driver manager, you should always use the `SQL_DRIVER_NOPROMPT` value for the `SQLDriverConnect` function's `DriverCompletion` parameter (the final parameter in the function call) when connecting to Vertica. Vertica's ODBC driver on Linux and UNIX platforms does not contain a UI, and therefore cannot prompt users for a password.
- On Windows client platforms, the ODBC driver can prompt users for connection information. See [Prompting Windows Users for Missing Connection Properties](#) for more information.
- If your database does not comply with your Vertica license agreement, your application receives a warning message in the return value of the `SQLConnect()` function. Always have your application examine this return value to see if it is `SQL_SUCCESS_WITH_INFO`. If it is, have your application extract and display the message to the user.

Enabling Native Connection Load Balancing in ODBC

Native connection load balancing helps spread the overhead caused by client connections on the hosts in the Vertica database. Both the server and the client must enable native connection load balancing in order for it to have an effect. If both have enabled it, then when the client initially connects to a host in the database, the host picks a host to handle the client connection from a list of the currently up hosts in the database, and informs the client which host it has chosen.

If the initially-contacted host did not choose itself to handle the connection, the client disconnects, then opens a second connection to the host selected by the first host. The

connection process to this second host proceeds as usual—if SSL is enabled, then SSL negotiations begin, otherwise the client begins the authentication process. See [About Native Connection Load Balancing](#) in the Administrator's Guide for details.

To enable native load balancing on your client, set the `ConnectionLoadBalance` connection parameter to true either in the DSN entry or in the connection string. The following example demonstrates connecting to the database several times with native connection load balancing enabled, and fetching the name of the node handling the connection from the `V_MONITOR.CURRENT_SESSION` system table.

```
// Demonstrate enabling native load connection balancing.
// Standard i/o library
#include <stdlib.h>
#include <iostream>
#include <assert.h>
// Only needed for Windows clients
// #include <windows.h>
// SQL include files that define data types and ODBC API
// functions
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>

using namespace std;
int main()
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hdlEnv; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    assert(SQL_SUCCEEDED(ret));

    // Set the ODBC version we are going to use to
    // 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    assert(SQL_SUCCEEDED(ret));

    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    assert(SQL_SUCCEEDED(ret));

    // Connect four times. If load balancing is on, client should
    // connect to different nodes.
    for (int x=1; x <= 4; x++) {

        // Connect to the database using SQLDriverConnect. Set
        // ConnectionLoadBalance to 1 (true) to enable load
        // balancing.
        cout << endl << "Connection attempt #" << x << "... ";
        const char *connStr = "DSN=VMart;ConnectionLoadBalance=1;"
            "UID=ExampleUser;PWD=password123";

        ret = SQLDriverConnect(hdlDbc, NULL, (SQLCHAR*)connStr, SQL_NTS,
            NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
    }
}
```

```
if(!SQL_SUCCEEDED(ret)) {
    cout << "failed. Exiting." << endl;
    exit(EXIT_FAILURE);
} else {
    cout << "succeeded" << endl;
}
// We're connected. Query the v_monitor.current_session table to
// find the name of the node we've connected to.

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
assert(SQL_SUCCEEDED(ret));

ret = SQLExecDirect( hdlStmt, (SQLCHAR*)"SELECT node_name FROM "
    "V_MONITOR.CURRENT_SESSION;", SQL_NTS );

if(SQL_SUCCEEDED(ret)) {
    // Bind variable to column in result set.
    SQLTCHAR node_name[256];
    ret = SQLBindCol(hdlStmt, 1, SQL_C_TCHAR, (SQLPOINTER)node_name,
        sizeof(node_name), NULL);
    while(SQL_SUCCEEDED(ret = SQLFetchScroll(hdlStmt, SQL_FETCH_NEXT,1))) {
        // Print the bound variables, which now contain the values from the
        // fetched row.
        cout << "Connected to node " << node_name << endl;
    }
}
// Free statement handle
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
cout << "Disconnecting." << endl;
ret = SQLDisconnect( hdlDbc );
assert(SQL_SUCCEEDED(ret));
}
// When done, free all of the handles to close them
// in an orderly fashion.
cout << endl << "Freeing handles..." << endl;
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
cout << "Done!" << endl;
exit(EXIT_SUCCESS);
}
```

Running the above example produces output similar to the following:

```
Connection attempt #1... succeeded
Connected to node v_vmart_node0001
Disconnecting.

Connection attempt #2... succeeded
Connected to node v_vmart_node0002
Disconnecting.

Connection attempt #3... succeeded
Connected to node v_vmart_node0003
Disconnecting.

Connection attempt #4... succeeded
```

```
Connected to node v_vmart_node0001
Disconnecting.

Freeing handles...
Done!
```

ODBC Connection Failover

If a client application attempts to connect to a host in the Vertica Analytics Platform cluster that is down, the connection attempt fails when using the default connection configuration. This failure usually returns an error to the user. The user must either wait until the host recovers and retry the connection or manually edit the connection settings to choose another host.

Due to Vertica Analytics Platform's distributed architecture, you usually do not care which database host handles a client application's connection. You can use the client driver's connection failover feature to prevent the user from getting connection errors when the host specified in the connection settings is unreachable. It gives you two ways to let the client driver automatically attempt to connect to a different host if the one specified in the connection parameters is unreachable:

- Configure your DNS server to return multiple IP addresses for a host name. When you use this host name in the connection settings, the client attempts to connect to the first IP address from the DNS lookup. If the host at that IP address is unreachable, the client tries to connect to the second IP, and so on until it either manages to connect to a host or it runs out of IP addresses.
- Supply a list of backup hosts for the client driver to try if the primary host you specify in the connection parameters is unreachable.

For both methods, the process of failover is transparent to the client application (other than specifying the list of backup hosts, if you choose to use the list method of failover). If the primary host is unreachable, the client driver automatically tries to connect to other hosts.

Failover only applies to the initial establishment of the client connection. If the connection breaks, the driver does not automatically try to reconnect to another host in the database.

Choosing a Failover Method

You usually choose to use one of the two failover methods. However, they do work together. If your DNS server returns multiple IP addresses and you supply a list of backup hosts, the client first tries all of the IPs returned by the DNS server, then the hosts in the backup list.

Note: If a host name in the backup host list resolves to multiple IP addresses, the client does not try all of them. It just tries the first IP address in the list.

The DNS method of failover centralizes the configuration client failover. As you add new nodes to your Vertica Analytics Platform cluster, you can choose to add them to the failover list by editing the DNS server settings. All client systems that use the DNS server to connect to Vertica Analytics Platform automatically use connection failover without having to change any settings. However, this method does require administrative access to the DNS server that all clients use to connect to the Vertica Analytics Platform cluster. This may not be possible in your organization.

Using the backup server list is easier than editing the DNS server settings. However, it decentralizes the failover feature. You may need to update the application settings on each client system if you make changes to your Vertica Analytics Platform cluster.

Using DNS Failover

To use DNS failover, you need to change your DNS server's settings to map a single host name to multiple IP addresses of hosts in your Vertica Analytics Platform cluster. You then have all client applications use this host name to connect to Vertica Analytics Platform.

You can choose to have your DNS server return as many IP addresses for the host name as you want. In smaller clusters, you may choose to have it return the IP addresses of all of the hosts in your cluster. However, for larger clusters, you should consider choosing a subset of the hosts to return. Otherwise there can be a long delay as the client driver tries unsuccessfully to connect to each host in a database that is down.

Using the Backup Host List

To enable backup list-based connection failover, your client application has to specify at least one IP address or host name of a host in the `BackupServerNode` parameter. The host name or IP can optionally be followed by a colon and a port number. If not supplied, the driver defaults to the standard Vertica port number (5433). To list multiple hosts, separate them by a comma.

The following example demonstrates setting the `BackupServerNode` connection parameter to specify additional hosts for the connection attempt. The connection string intentionally has a non-existent node, so that the initial connection fails. The client driver has to resort to trying the backup hosts to establish a connection to Vertica.

```
// Demonstrate using connection failover.
// Standard i/o library
#include <stdlib.h>;
#include <iostream>;
#include <assert.h>;

// Only needed for Windows clients
// #include <windows.h>;

// SQL include files that define data types and ODBC API
// functions
#include <sql.h>;
#include <sqlext.h>;
#include <sqltypes.h>;

using namespace std;

int main()
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hdlEnv; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    assert(SQL_SUCCEEDED(ret));

    // Set the ODBC version we are going to use to
    // 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    assert(SQL_SUCCEEDED(ret));

    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    assert(SQL_SUCCEEDED(ret));

    /* DSN for this connection specifies a bad node, and good backup nodes:
    [VMartBadNode]
    Description=VMart Vertica Database
    Driver=/opt/vertica/lib64/libverticaodbc.so
    Database=VMart
    Servername=badnode.example.com
    BackupServerNode=v_vmart_node0002.example.com,v_vmart_node0003.example.com
    */

    // Connect to the database using SQLConnect
    cout <<< "Connecting to database." <<< endl;
    const char *dsnName = "VMartBadNode"; // Name of the DSN
    const char *userID = "ExampleUser"; // Username
    const char *passwd = "password123"; // password
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
        SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
        (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        cout <<< "Could not connect to database." <<< endl;
        exit(EXIT_FAILURE);
    } else {
        cout <<< "Connected to database." <<< endl;
    }
    // We're connected. Query the v_monitor.current_session table to
```

```
// find the name of the node we've connected to.

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
assert(SQL_SUCCEEDED(ret));

ret = SQLExecDirect( hdlStmt, (SQLCHAR*)"SELECT node_name FROM "
    "v_monitor.current_session;", SQL_NTS );

if(SQL_SUCCEEDED(ret)) {
    // Bind variable to column in result set.
    SQLTCHAR node_name[256];
    ret = SQLBindCol(hdlStmt, 1, SQL_C_TCHAR, (SQLPOINTER)node_name,
        sizeof(node_name), NULL);
    while(SQL_SUCCEEDED(ret = SQLFetchScroll(hdlStmt, SQL_FETCH_NEXT,1))) {
        // Print the bound variables, which now contain the values from the
        // fetched row.
        cout &&& "Connected to node " &&& node_name &&& endl;
    }
}

cout &&& "Disconnecting." &&& endl;
ret = SQLDisconnect( hdlDbc );
assert(SQL_SUCCEEDED(ret));

// When done, free all of the handles to close them
// in an orderly fashion.
cout &&& endl &&& "Freeing handles..." &&& endl;
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
cout &&& "Done!" &&& endl;
exit(EXIT_SUCCESS);
}
```

When run, the example's output on the system console is similar to the following:

```
Connecting to database.
Connected to database.
Connected to node v_vmart_node0002
Disconnecting.

Freeing handles...
Done!
```

Notice that the connection was made to the first node in the backup list (node 2).

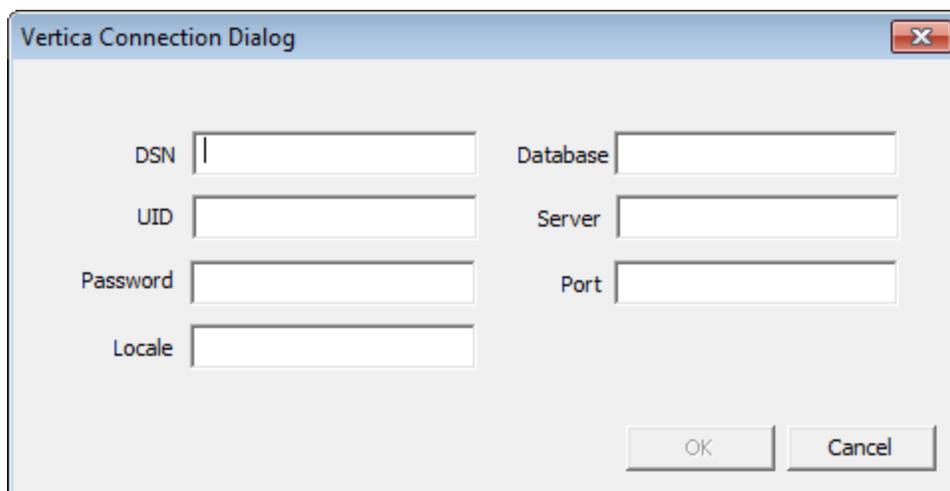
Note: When native connection load balancing is enabled, the additional servers specified in the BackupServerNode connection parameter are only used for the initial connection to a Vertica host. If host redirects the client to another host in the database cluster to handle its connection request, the second connection does not use the backup node list. This is rarely an issue, since native connection load balancing is aware of which nodes are currently up in the database. See [Enabling Native Connection Load Balancing in ODBC](#)

Prompting Windows Users for Missing Connection Properties

The Vertica Windows ODBC driver can prompt the user for connection information if required information is missing. The driver displays the Vertica Connection Dialog if the client application calls `SQLDriverConnect` to connect to Vertica and either of the following is true:

- the `DriverCompletion` property is set to `SQL_DRIVER_PROMPT`.
- the `DriverCompletion` property is set to `SQL_DRIVER_COMPLETE` or `SQL_DRIVER_COMPLETE_REQUIRED` and the connection string or DSN being used to connect is missing the server, database, or port information.

If either of the above conditions are true, the driver displays a Vertica Connection Dialog to the user to prompt for connection information.



The screenshot shows a standard Windows dialog box titled "Vertica Connection Dialog". It features a close button (X) in the top right corner. The dialog contains two columns of text boxes. The left column includes fields for "DSN", "UID", "Password", and "Locale". The right column includes fields for "Database", "Server", and "Port". At the bottom center, there are two buttons: "OK" and "Cancel".

The dialog has all of the property values supplied in the connection string or DSN filled in.

Note: Your connection string at least needs to specify Vertica as the driver, otherwise Windows will not know to use the Vertica ODBC driver to try to open the connection.

The required fields on the connection dialog are Database, UID, Server, and Port. Once these are filled in, the form enables the **OK** button.

If the user clicks **Cancel** on the dialog, the `SQLDriverConnect` function call returns `SQL_NO_DATA` immediately, without attempting to connect to Vertica. If the user supplies incomplete or incorrect information for the connection, the connection function returns `SQL_ERROR` after the connection attempt fails.

Note: If the `DriverCompletion` property of the `SQLDriverConnect` function call is `SQL_DRIVER_NOPROMPT`, the ODBC driver immediately returns a `SQL_ERROR` indicating that it cannot connect because not enough information has been supplied and the driver is not allowed to prompt the user for the missing information.

Prompting Windows Users for Passwords

If the connection string or DSN supplied to the `SQLDriverConnect` function that client applications call to connect to Vertica lacks any of the required connection properties needed to connect, the Vertica's Windows ODBC driver opens a dialog box to prompt the user to enter the missing information (see [Prompting Windows Users for Missing Connection Properties](#)). The user's password is not normally considered a required connection property, since Vertica user accounts may not have a password. If the password property is missing, the ODBC driver still tries to connect to Vertica without supplying a password.

You can use the `PromptOnNoPassword` DSN parameter to force ODBC driver to treat the password as a required connection property. This parameter is useful if you do not want to store passwords in DSN entries. Passwords saved in DSN entries are insecure, since they are stored as clear text in the Windows registry and therefore visible to other users on the same system.

There are two other factors which also decide whether the ODBC driver displays the Vertica Connection Dialog. These are (in order of priority):

- The `SQLDriverConnect` function call's `DriverCompletion` parameter.
- Whether the DSN or connection string contain a password

The following table shows how the `PromptOnNoPassword` DSN parameter, the `DriverCompletion` parameter of the `SQLDriverConnect` function, and whether the DSN or connection string contains a password interact to control whether the Vertica Connection dialog appears.

PromptOnNoPassword Setting	DriverCompletion Value	DSN or Connection String Contains Password?	Vertica Connection Dialog Displays?	Notes
any value	SQL_DRIVER_PROMPT	any case	Yes	This <code>DriverCompletion</code> value forces the

PromptOnNoPassword Setting	DriverCompletion Value	DSN or Connection String Contains Password?	Vertica Connection Dialog Displays?	Notes
				dialog to always appear, even if all required connection properties are supplied.
any value	SQL_DRIVER_NOPROMPT	any case	No	This DriverCompletion value always prevents the dialog from appearing.
any value	SQL_DRIVER_COMPLETE	Yes	No	Connection dialog displays if another required connection property is missing.
true	SQL_DRIVER_COMPLETE	No	Yes	
false (default)	SQL_DRIVER_COMPLETE	No	No	Connection dialog displays if another required connection property is missing.

The following example code demonstrates using the PromptOnNoPassword DSN parameter along with a system DSN.

```
wstring connectString = L"DSN=VerticaDSN;PromptOnNoPassword=1;";
retcode = SQLDriverConnect(
    hdbc,
```

```
0,  
(SQLWCHAR*)connectString.c_str(),  
connectString.length(),  
OutConnStr,  
255,  
&OutConnStrLen,  
SQL_DRIVER_COMPLETE );
```

No Password Entry vs. Empty Passwords

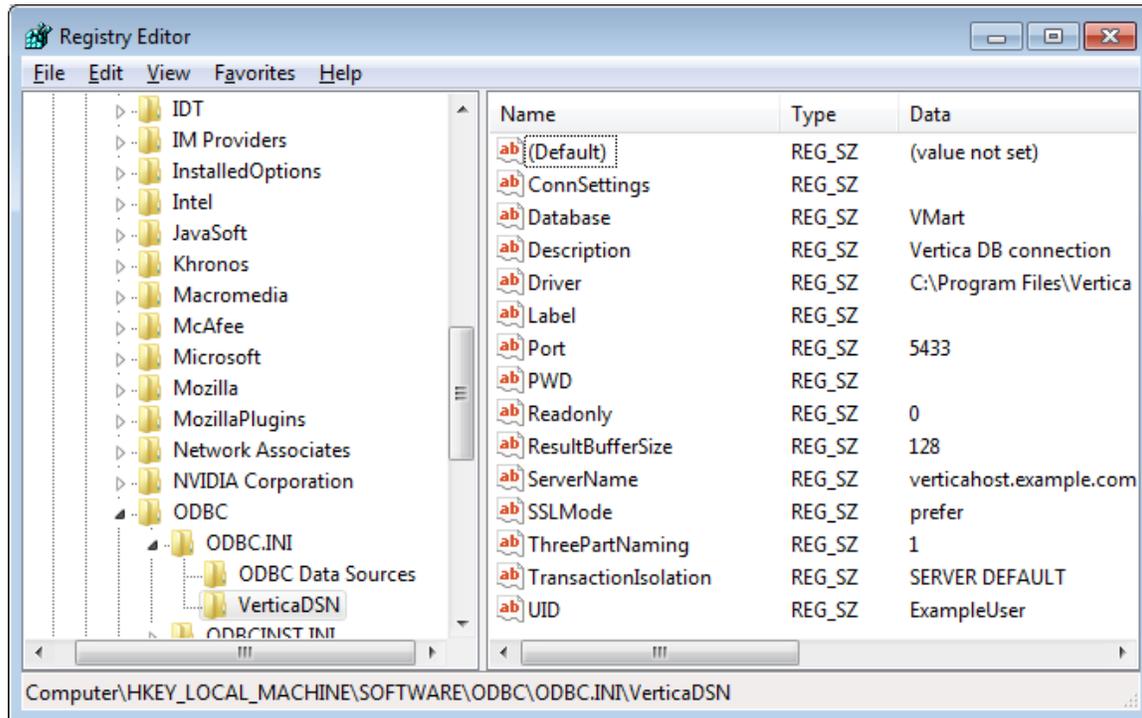
There is a difference between not having a password property in the connection string or DSN and having an empty password. The PromptOnNoPassword DSN parameter only has an effect if the connection string or DSN does not have a PWD property (which holds the user's password). If it does, even if it is empty, PromptOnNoPassword will not prompt the Windows ODBC driver to display the Vertica Connection Dialog.

This difference can cause confusion if you are using a DSN to provide the properties for your connection. Once you enter a password for a DSN connection in the Windows ODBC Manager and save it, Windows adds a PWD property to the DSN definition in the registry. If you later delete the password, the PWD property remains in the DSN definition—value is just set to an empty string. The PWD property is created even if you just use the Test button on the ODBC Manager dialog to test the DSN and later clear it before saving the DSN.

Once the password has been set, the only way to remove the PWD property from the DSN definition is to delete it using the Windows Registry Editor:

1. On the Windows Start menu, click Run.
2. In the Run dialog, type regedit, then click OK.
3. In the Registry Editor window, click Edit > Find (or press Ctrl+F).
4. In the Find window, enter the name of the DSN whose PWD property you want to delete and click OK.

5. If find operation did not locate a folder under the ODBC.INI folder, click Edit > Find Next (or press F3) until the folder matching your DSN's name is highlighted.



6. Select the PWD entry and press Delete.
7. Click Yes to confirm deleting the value.

The DSN now does not have a PWD property and can trigger the connection dialog to appear when used along with `PromptOnNoPassword=true` and `DriverConnect=SQL_DRIVER_COMPLETE`.

Setting the Locale for ODBC Sessions

Vertica provides three ways to set the locale for an ODBC session:

- Specify the locale for all connections made using the DSN:
 - On Linux and other UNIX-like platforms: [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)

- On Windows platforms, set the locale in the ODBC DSN configuration editor's Locale field on the Server Settings tab. See [Creating an ODBC DSN for Windows Clients](#) for detailed information.
- Set the Locale connection parameter in the connection string in `SQLDriverConnect()` function.

For example:

```
SQLDriverConnect(conn, NULL, (SQLCHAR*)"DSN=Vertica;Locale=en_GB@collation=binary", SQL_NTS, szConnOut, sizeof(szConnOut), &iAvailable, SQL_DRIVER_NOPROMPT)
```

- Use the `SQLSetConnectAttr()` method with the `SQL_ATTR_VERTICA_LOCALE` constant and specify the ICU string as the attribute value. See the example below.

Notes

- Having the client system use a non-Unicode locale (such as setting `LANG=C` on Linux platforms) and using a Unicode locale for the connection to Vertica can result in errors such as "(10170) String data right truncation on data from data source." If data received from Vertica isn't in UTF-8 format. The driver allocates string memory based on the system's locale setting, and non-UTF-8 data can trigger an overrun. You can avoid these errors by always using a Unicode locale on the client system.

If you specify a locale either in the connection string or in the DSN, the call to the connection function returns `SQL_SUCCESS_WITH_INFO` on a successful connection, with messages about the state of the locale.

- ODBC applications can be in either ANSI or Unicode mode:
 - If Unicode, the encoding used by ODBC is UCS-2.
 - If ANSI, the data must be in single-byte ASCII, which is compatible with UTF-8 on the database server.

The ODBC driver converts UCS-2 to UTF-8 when passing to the Vertica server and converts data sent by the Vertica server from UTF-8 to UCS-2.

- If the end-user application is not already in UCS-2, the application is responsible for converting the input data to UCS-2, or unexpected results could occur. For example:

- On non-UCS-2 data passed to ODBC APIs, when it is interpreted as UCS-2, it could result in an invalid UCS-2 symbol being passed to the APIs, resulting in errors.
- Or the symbol provided in the alternate encoding could be a valid UCS-2 symbol; in this case, incorrect data is inserted into the database.

ODBC applications should set the correct server session locale using `SQLSetConnectAttr` (if different from database-wide setting) in order to set the proper collation and string functions behavior on server.

The following example code demonstrates setting the locale using both the connection string and through the `SQLSetConnectAttr()` function.

```
// Standard i/o library
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// SQL include files that define data types and ODBC API
// functions
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
// Vertica-specific definitions. This include file is located as
// /opt/vertica/include on database hosts.
#include <verticaodbc.h>
int main()
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hdlEnv; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Set the ODBC version we are going to use to 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC 3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application version to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated Database handle.\n");
    }
}
```

```
}
// Connect to the database using SQLDriverConnect
printf("Connecting to database.\n");
// Set the locale to English in Great Britain.
const char *connStr = "DSN=ExampleDB;locale=en_GB;"
    "UID=dbadmin;PWD=password123";
ret = SQLDriverConnect(hdlDbc, NULL, (SQLCHAR*)connStr, SQL_NTS,
    NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}
// Get the Locale
char locale[256];
SQLGetConnectAttr(hdlDbc, SQL_ATTR_VERTICA_LOCALE, locale, sizeof(locale),
    0);
printf("Locale is set to: %s\n", locale);
// Set the locale to a new value
const char* newLocale = "en_GB";
SQLSetConnectAttr(hdlDbc, SQL_ATTR_VERTICA_LOCALE, (SQLCHAR*)newLocale,
    SQL_NTS);

// Get the Locale again
SQLGetConnectAttr(hdlDbc, SQL_ATTR_VERTICA_LOCALE, locale, sizeof(locale),
    0);
printf("Locale is now set to: %s\n", locale);
// When done, free all of the handles to close them
// in an orderly fashion.
printf("Disconnecting and freeing handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting from database. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

AUTOCOMMIT and ODBC Transactions

The AUTOCOMMIT connection attribute controls whether INSERT, ALTER, COPY and other data-manipulation statements are automatically committed after they complete. By default, AUTOCOMMIT is enabled—all statements are committed after they execute. This is often not the best setting to use, since it is less efficient. Also, you often want to control whether a set of statements are committed as a whole, rather than have each individual statement committed. For example, you may only want to commit a series of inserts if all of the inserts succeed. With AUTOCOMMIT disabled, you can roll back the transaction if one of the statements fail.

If AUTOCOMMIT is on, the results of statements are committed immediately after they are executed. You cannot roll back a statement executed in AUTOCOMMIT mode.

For example, when AUTOCOMMIT is on, the following single INSERT statement is automatically committed:

```
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"INSERT INTO customers VALUES(500,"  
    "Smith, Sam', '123-456-789');", SQL_NTS);
```

If AUTOCOMMIT is off, you need to manually commit the transaction after executing a statement. For example:

```
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"INSERT INTO customers VALUES(500,"  
    "Smith, Sam', '123-456-789');", SQL_NTS);  
// Other inserts and data manipulations  
// Commit the statements(s)  
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
```

The inserted row is only committed when you call `SQLEndTran()`. You can roll back the INSERT and other statements at any point before committing the transaction.

Note: Prepared statements cache the AUTOCOMMIT setting when you create them using `SQLPrepare()`. Later changing the connection's AUTOCOMMIT setting has no effect on the AUTOCOMMIT settings of previously created prepared statements. See [Using Prepared Statements](#) for details.

The following example demonstrates turning off AUTOCOMMIT, executing an insert, then manually committing the transaction.

```
// Some standard headers  
#include <stdio.h>  
#include <stdlib.h>  
// Only needed for Windows clients  
// #include <windows.h>  
// Standard ODBC headers  
#include <sql.h>  
#include <sqltypes.h>  
#include <sqlext.h>  
int main()  
{  
    // Set up the ODBC environment  
    SQLRETURN ret;  
    SQLHENV hdlEnv;  
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);  
    if(!SQL_SUCCEEDED(ret)) {  
        printf("Could not allocate a handle.\n");  
        exit(EXIT_FAILURE);  
    } else {  
        printf("Allocated an environment handle.\n");  
    }  
    // Tell ODBC that the application uses ODBC 3.  
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,  
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);  
    if(!SQL_SUCCEEDED(ret)) {  
        printf("Could not set application version to ODBC3.\n");  
        exit(EXIT_FAILURE);  
    }  
}
```

```
} else {
    printf("Set application to ODBC 3.\n");
}
// Allocate a database handle.
SQLHDBC hdlDbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate database handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated Database handle.\n");
}
// Connect to the database
printf("Connecting to database.\n");
const char *dsnName = "ExampleDB";
const char *userID = "dbadmin";
const char *passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
    SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
    (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}
// Get the AUTOCOMMIT state
SQLINTEGER autoCommitState;
SQLGetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, &autoCommitState, 0, NULL);
printf("Autocommit is set to: %d\n", autoCommitState);

// Disable AUTOCOMMIT
printf("Disabling autocommit.\n");
ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF,
    SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not disable autocommit.\n");
    exit(EXIT_FAILURE);
}

// Get the AUTOCOMMIT state again
SQLGetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, &autoCommitState, 0, NULL);
printf("Autocommit is set to: %d\n", autoCommitState);

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Create a table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
    SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
    "(CustID int, CustName varchar(100), Phone_Number char(15));",
    SQL_NTS);

// Insert a single row.
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"INSERT INTO customers VALUES(500,"
```

```
    ""Smith, Sam', '123-456-789');", SQL_NTS);
if(!ISQL_SUCCEEDED(ret)) {
    printf("Could not perform single insert.\n");
} else {
    printf("Performed single insert.\n");
}

// Need to commit the transaction before closing, since autocommit is
// disabled. Otherwise SQLDisconnect returns an error.
printf("Committing transaction.\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!ISQL_SUCCEEDED(ret)) {
    printf("Error committing transaction.\n");
    exit(EXIT_FAILURE);
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect(hdlDbc);
if(!ISQL_SUCCEEDED(ret)) {
    printf("Error disconnecting from database. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

Running the above code results in the following output:

```
Allocated an environment handle.
Set application to ODBC 3.
Allocated Database handle.
Connecting to database.
Connected to database.
Autocommit is set to: 1
Disabling autocommit.
Autocommit is set to: 0
Performed single insert.
Committing transaction.
Free handles.
```

Note: You can also disable AUTOCOMMIT in the ODBC connection string. See [Setting DSN Connection Properties](#) for more information.

Retrieving Data Through ODBC

To retrieve data through ODBC, you execute a query that returns a result set ([SELECT](#), for example), then retrieve the results using one of two methods:

- Use the `SQLFetch()` function to retrieve a row of the result set, then access column values in the row by calling `SQLGetData()`.
- Use the `SQLBindColumn()` function to bind a variable or array to a column in the result set, then call `SQLExtendedFetch()` or `SQLFetchScroll()` to read a row of the result set and insert its values into the variable or array.

In both methods you loop through the result set until you either reach the end (signaled by the `SQL_NO_DATA` return status) or encounter an error.

Note: Vertica supports one cursor per connection. Attempting to use more than one cursor per connection will result in an error. For example, you receive an error if you execute a statement while another statement has a result set open.

The following code example demonstrates retrieving data from Vertica by:

1. Connecting to the database.
2. Executing a `SELECT` statement that returns the IDs and names of all tables.
3. Binds two variables to the two columns in the result set.
4. Loops through the result set, printing the ids and name values.

```
// Demonstrate running a query and getting results by querying the tables
// system table for a list of all tables in the current schema.
// Some standard headers
#include <stdlib.h>
#include <sstream>
#include <iostream>
#include <assert.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
// Use std namespace to make output easier
using namespace std;
// Helper function to print SQL error messages.
template <typename HandleT>
void reportError(int handleTypeEnum, HandleT hdl)
{
    // Get the status records.
    SQLSMALLINT i, MsgLen;
    SQLRETURN ret2;
    SQLCHAR SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER NativeError;
    i = 1;
    cout << endl;
    while ((ret2 = SQLGetDiagRec(handleTypeEnum, hdl, i, SqlState, &NativeError,
        Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA) {
        cout << "error record #" << i++ << endl;
    }
}
```

```
        cout << "sqlstate: " << SqlState << endl;
        cout << "detailed msg: " << Msg << endl;
        cout << "native error code: " << NativeError << endl;
    }
}
int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    assert(SQL_SUCCEEDED(ret));
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    assert(SQL_SUCCEEDED(ret));
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    assert(SQL_SUCCEEDED(ret));
    // Connect to the database
    cout << "Connecting to database." << endl;
    const char* dsnName = "ExampleDB";
    const char* userID = "dbadmin";
    const char* passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
        SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
        (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        cout << "Could not connect to database" << endl;
        reportError<SQLHDBC>(SQL_HANDLE_DBC, hdlDbc);
        exit(EXIT_FAILURE);
    } else {
        cout << "Connected to database." << endl;
    }

    // Set up a statement handle
    SQLHSTMT hdlStmt;
    SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
    assert(SQL_SUCCEEDED(ret));

    // Execute a query to get the names and IDs of all tables in the schema
    // search p[ath] (usually public).
    ret = SQLExecDirect( hdlStmt, (SQLCHAR*)"SELECT table_id, table_name "
        "FROM tables ORDER BY table_name", SQL_NTS );

    if(!SQL_SUCCEEDED(ret)) {
        // Report error and go no further if statement failed.
        cout << "Error executing statement." << endl;
        reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);
        exit(EXIT_FAILURE);
    } else {

        // Query succeeded, so bind two variables to the two columns in the
        // result set,
        cout << "Fetching results..." << endl;
        SQLBIGINT table_id; // Holds the ID of the table.
        SQLTCHAR table_name[256]; // buffer to hold name of table
```

```
ret = SQLBindCol(hdlStmt, 1, SQL_C_SBIGINT, (SQLPOINTER)&table_id,
    sizeof(table_id), NULL);
ret = SQLBindCol(hdlStmt, 2, SQL_C_TCHAR, (SQLPOINTER)table_name,
    sizeof(table_name), NULL);

// Loop through the results,
while( SQL_SUCCEEDED(ret = SQLFetchScroll(hdlStmt, SQL_FETCH_NEXT,1))) {
    // Print the bound variables, which now contain the values from the
    // fetched row.
    cout << table_id << " | " << table_name << endl;
}

// See if loop exited for reasons other than running out of data
if (ret != SQL_NO_DATA) {
    // Exited for a reason other than no more data... report the error.
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt);
}
}

// Clean up by shutting down the connection
cout << "Free handles." << endl;
ret = SQLDisconnect( hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    cout << "Error disconnecting. Transaction still open?" << endl;
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

Running the example code in the vmart database produces output similar to this:

```
Connecting to database.
Connected to database.
Fetching results...
45035996273970908 | call_center_dimension
45035996273970836 | customer_dimension
45035996273972958 | customers
45035996273970848 | date_dimension
45035996273970856 | employee_dimension
45035996273970868 | inventory_fact
45035996273970904 | online_page_dimension
45035996273970912 | online_sales_fact
45035996273970840 | product_dimension
45035996273970844 | promotion_dimension
45035996273970860 | shipping_dimension
45035996273970876 | store_dimension
45035996273970894 | store_orders_fact
45035996273970880 | store_sales_fact
45035996273972806 | t
45035996273970852 | vendor_dimension
45035996273970864 | warehouse_dimension
Free handles.
```

Loading Data Through ODBC

A primary task for many client applications is loading data into the Vertica database. There are several different ways to insert data using ODBC, which are covered by the topics in this section.

Using a Single Row Insert

The easiest way to load data into Vertica is to run an INSERT SQL statement using the `SQLExecuteDirect` function. However this method is limited to inserting a single row of data.

```
ret = SQLExecDirect(hstmt, (SQLTCHAR*)"INSERT into Customers values"  
    "(1,'abcd','efgh','1')", SQL_NTS);
```

Using Prepared Statements

Vertica supports using server-side prepared statements with both ODBC and JDBC. Prepared statements let you define a statement once, and then run it many times with different parameters. The statement you want to execute contains placeholders instead of parameters. When you execute the statement, you supply values for each placeholder.

Placeholders are represented by question marks (?) as in the following example query:

```
SELECT * FROM public.inventory_fact WHERE product_key = ?
```

Server-side prepared statements are useful for:

- Optimizing queries. Vertica only needs to parse the statement once.
- Preventing SQL injection attacks. A SQL injection attack occurs when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly run. Since a prepared statement is parsed separately from the input data, there is no chance the data can be accidentally executed by the database.
- Binding direct variables to return columns. By pointing to data structures, the code doesn't have to perform extra transformations.

The following example demonstrates a using a prepared statement for a single insert.

```
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
// Some constants for the size of the data to be inserted.
#define CUST_NAME_LEN 50
#define PHONE_NUM_LEN 15
#define NUM_ENTRIES 4
int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char *userID = "dbadmin";
    const char *passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
        SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
        (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not connect to database.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Connected to database.\n");
    }

    // Disable AUTOCOMMIT
    printf("Disabling autocommit.\n");
    ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF,
        SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not disable autocommit.\n");
        exit(EXIT_FAILURE);
    }
}
```

```
// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
    SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
    "(CustID int, CustName varchar(100), Phone_Number char(15));",
    SQL_NTS);

// Set up a bunch of variables to be bound to the statement
// parameters.

// Create the prepared statement. This will insert data into the
// table we created above.
printf("Creating prepared statement\n");
ret = SQLPrepare (hdlStmt, (SQLTCHAR*)"INSERT INTO customers (CustID, "
    "CustName, Phone_Number) VALUES(?,?,?)", SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not create prepared statement\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
    exit(EXIT_FAILURE);
} else {
    printf("Created prepared statement.\n");
}
SQLINTEGER custID = 1234;
SQLCHAR custName[100] = "Fein, Fredrick";
SQLVARCHAR phoneNum[15] = "555-123-6789";
SQLLEN strFieldLen = SQL_NTS;
SQLLEN custIDLen = 0;
// Bind the data arrays to the parameters in the prepared SQL
// statement
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
    0, 0, &custID, 0, &custIDLen);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custID array\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
    exit(EXIT_FAILURE);
} else {
    printf("Bound custID to prepared statement\n");
}
// Bind CustNames
SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
    50, 0, (SQLPOINTER)custName, 0, &strFieldLen);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custNames\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
    exit(EXIT_FAILURE);
} else {
    printf("Bound custName to prepared statement\n");
}
// Bind phoneNums
SQLBindParameter(hdlStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
```

```
    15, 0, (SQLPOINTER)phoneNum, 0, &strFieldLen);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind phoneNums\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNum to prepared statement\n");
}
// Execute the prepared statement.
printf("Running prepared statement...");
ret = SQLExecute(hdlStmt);
if(!SQL_SUCCEEDED(ret)) {
    printf("not successful!\n");
} else {
    printf("successful.\n");
}

// Done with batches, commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not commit transaction\n");
} else {
    printf("Committed transaction\n");
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

Using Batch Inserts

You use batch inserts to insert chunks of data into the database. By breaking the data into batches, you can monitor the progress of the load by receiving information about any rejected rows after each batch is loaded. To perform a batch load through ODBC, you typically use a prepared statement with the parameters bound to arrays that contain the data to be loaded. For each batch, you load a new set of data into the arrays then execute the prepared statement.

When you perform a batch load, Vertica uses a [COPY](#) statement to load the data. Each additional batch you load uses the same COPY statement. The statement remains open until

you end the transaction, close the cursor for the statement, or execute a non-INSERT statement.

Using a single COPY statement for multiple batches improves batch loading efficiency by:

- reducing the overhead of inserting individual batches
- combining individual batches into larger ROS containers

Note: If the database connection has AUTOCOMMIT enabled, then the transaction is automatically committed after each batch insert statement which closes the COPY statement. Leaving AUTOCOMMIT enabled makes your batch load much less efficient, and can cause added overhead in your database as all of the smaller loads are consolidated.

Even though Vertica uses a single COPY statement to insert multiple batches within a transaction, you can locate which (if any) rows were rejected due to invalid row formats or data type issues after each batch is loaded. See [Finding the Number of Accepted Rows](#) for details.

Note: While you can find rejected rows during the batch load transaction, other types of errors (such as running out of disk space or a node shutdown that makes the database unsafe) are only reported when the COPY statement ends.

Since the batch loads share a COPY statement, errors in one batch can cause earlier batches in the same transaction to be rolled back.

Batch Insert Steps

The steps your application needs to take in order to perform an ODBC Batch Insert are:

1. Connect to the database.
2. Disable autocommit for the connection.
3. Create a prepared statement that inserts the data you want to load.
4. Bind the parameters of the prepared statement to arrays that will contain the data you want to load.
5. Populate the arrays with the data for your batches.
6. Execute the prepared statement.
7. Optionally, check the results of the batch load to find rejected rows.

8. Repeat the previous three steps until all of the data you want to load is loaded.
9. Commit the transaction.
10. Optionally, check the results of the entire batch transaction.

The following example code demonstrates a simplified version of the above steps.

```
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
int main()
{
    // Number of data rows to insert
    const int NUM_ENTRIES = 4;

    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated Database handle.\n");
    }
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char *userID = "dbadmin";
    const char *passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
        SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
```

```
(SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}

// Disable AUTOCOMMIT
printf("Disabling autocommit.\n");
ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF,
    SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not disable autocommit.\n");
    exit(EXIT_FAILURE);
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Create a table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
    SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
    "(CustID int, CustName varchar(100), Phone_Number char(15));",
    SQL_NTS);

// Create the prepared statement. This will insert data into the
// table we created above.
printf("Creating prepared statement\n");
ret = SQLPrepare (hdlStmt, (SQLCHAR*)"INSERT INTO customers (CustID, "
    "CustName, Phone_Number) VALUES(?,?,?)", SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not create prepared statement\n");
    exit(EXIT_FAILURE);
} else {
    printf("Created prepared statement.\n");
}

// This is the data to be inserted into the database.
SQLCHAR custNames[50] = { "Allen, Anna", "Brown, Bill", "Chu, Cindy",
    "Dodd, Don" };
SQLINTEGER custIDs[] = { 100, 101, 102, 103};
SQLCHAR phoneNums[15] = {"1-617-555-1234", "1-781-555-1212",
    "1-508-555-4321", "1-617-555-4444"};
// Bind the data arrays to the parameters in the prepared SQL
// statement. First is the custID.
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
    0, 0, (SQLPOINTER)custIDs, sizeof(SQLINTEGER), NULL);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custID array\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound CustIDs array to prepared statement\n");
}
// Bind CustNames
ret = SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
    50, 0, (SQLPOINTER)custNames, 50, NULL);
if(!SQL_SUCCEEDED(ret)) {
```

```
    printf("Could not bind custNames\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound CustNames array to prepared statement\n");
}
// Bind phoneNums
ret = SQLBindParameter(hdlStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
    15, 0, (SQLPOINTER)phoneNums, 15, NULL);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind phoneNums\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNums array to prepared statement\n");
}
// Tell the ODBC driver how many rows we have in the
// array.
ret = SQLSetStmtAttr( hdlStmt, SQL_ATTR_PARAMSET_SIZE,
    (SQLPOINTER)NUM_ENTRIES, 0);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind set parameter size\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNums array to prepared statement\n");
}

// Add multiple batches to the database. This just adds the same
// batch of data four times for simplicity's sake. Each call adds
// the 4 rows into the database.
for (int batchLoop=1; batchLoop<=5; batchLoop++) {
    // Execute the prepared statement, loading all of the data
    // in the arrays.
    printf("Adding Batch #%d...", batchLoop);
    ret = SQLExecute(hdlStmt);
    if(!SQL_SUCCEEDED(ret)) {
        printf("not successful!\n");
    } else {
        printf("successful.\n");
    }
}
// Done with batches, commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not commit transaction\n");
} else {
    printf("Committed transaction\n");
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

The result of running the above code is shown below.

```
Allocated an environment handle.  
Set application to ODBC 3.  
Allocated Database handle.  
Connecting to database.  
Connected to database.  
Creating prepared statement  
Created prepared statement.  
Bound CustIDs array to prepared statement  
Bound CustNames array to prepared statement  
Bound phoneNums array to prepared statement  
Adding Batch #1...successful.  
Adding Batch #2...successful.  
Adding Batch #3...successful.  
Adding Batch #4...successful.  
Adding Batch #5...successful.  
Committing transaction  
Committed transaction  
Free handles.
```

The resulting table looks like this:

```
=> SELECT * FROM customers;  
CustID | CustName | Phone_Number  
-----+-----+-----  
  100 | Allen, Anna | 1-617-555-1234  
  101 | Brown, Bill | 1-781-555-1212  
  102 | Chu, Cindy | 1-508-555-4321  
  103 | Dodd, Don | 1-617-555-4444  
  100 | Allen, Anna | 1-617-555-1234  
  101 | Brown, Bill | 1-781-555-1212  
  102 | Chu, Cindy | 1-508-555-4321  
  103 | Dodd, Don | 1-617-555-4444  
  100 | Allen, Anna | 1-617-555-1234  
  101 | Brown, Bill | 1-781-555-1212  
  102 | Chu, Cindy | 1-508-555-4321  
  103 | Dodd, Don | 1-617-555-4444  
  100 | Allen, Anna | 1-617-555-1234  
  101 | Brown, Bill | 1-781-555-1212  
  102 | Chu, Cindy | 1-508-555-4321  
  103 | Dodd, Don | 1-617-555-4444  
  100 | Allen, Anna | 1-617-555-1234  
  101 | Brown, Bill | 1-781-555-1212  
  102 | Chu, Cindy | 1-508-555-4321  
  103 | Dodd, Don | 1-617-555-4444  
(20 rows)
```

Note: An input parameter bound with the SQL_C_NUMERIC data type uses the default numeric precision (37) and the default scale (0) instead of the precision and scale set by the SQL_NUMERIC_STRUCT input value. This behavior adheres to the ODBC standard. If you do not want to use the default precision and scale, use `SQLSetDescField()` or `SQLSetDescRec()` to change them in the statement's attributes.

Tracking Load Status (ODBC)

After loading a batch of data, your client application can get the number of rows that were processed and find out whether each row was accepted or rejected.

Finding the Number of Accepted Rows

To get the number of rows processed by a batch, you add an attribute named `SQL_ATTR_PARAMS_PROCESSED_PTR` to the statement object that points to a variable to receive the number rows:

```
SQLULEN rowsProcessed;  
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &rowsProcessed, 0);
```

When your application calls `SQLExecute()` to insert the batch, the Vertica ODBC driver saves the number of rows that it processed (which is not necessarily the number of rows that were successfully inserted) in the variable you specified in the `SQL_ATTR_PARAMS_PROCESSED_PTR` statement attribute.

Finding the Accepted and Rejected Rows

Your application can also set a statement attribute named `SQL_ATTR_PARAM_STATUS_PTR` that points to an array where the ODBC driver can store the result of inserting each row:

```
SQLUSMALLINT rowResults[ NUM_ENTRIES ];  
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAM_STATUS_PTR, rowResults, 0);
```

This array must be at least as large as the number of rows being inserted in each batch.

When your application calls `SQLExecute` to insert a batch, the ODBC driver populates the array with values indicating whether each row was successfully inserted (`SQL_PARAM_SUCCESS` or `SQL_PARAM_SUCCESS_WITH_INFO`) or encountered an error (`SQL_PARAM_ERROR`).

The following example expands on the example shown in [Using Batch Inserts](#) to include reporting the number of rows processed and the status of each row inserted.

```
// Some standard headers  
#include <stdio.h>  
#include <stdlib.h>  
// Only needed for Windows clients  
// #include <windows.h>  
// Standard ODBC headers  
#include <sql.h>  
#include <sqltypes.h>  
#include <sqlext.h>  
// Helper function to print SQL error messages.  
template <typename HandleT>  
void reportError(int handleTypeEnum, HandleT hdl)  
{  
    // Get the status records.  
    SQLSMALLINT i, MsgLen;  
    SQLRETURN ret2;  
    SQLCHAR SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];  
    SQLINTEGER NativeError;  
    i = 1;  
    printf("\n");  
    while ((ret2 = SQLGetDiagRec(handleTypeEnum, hdl, i, SqlState, &NativeError,  
        Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA) {  
        printf("error record %d\n", i);  
        printf("sqlstate: %s\n", SqlState);  
        printf("detailed msg: %s\n", Msg);  
        printf("native error code: %d\n\n", NativeError);  
        i++;  
    }  
}  
int main()  
{  
    // Number of data rows to insert
```

```
const int NUM_ENTRIES = 4;

SQLRETURN ret;
SQLHENV hdlEnv;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate a handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated an environment handle.\n");
}
ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
(SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not set application version to ODBC3.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Set application to ODBC 3.\n");
}
SQLHDBC hdlDbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate database handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated Database handle.\n");
}
// Connect to the database
printf("Connecting to database.\n");
const char *dsnName = "ExampleDB";
const char *userID = "dbadmin";
const char *passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
(SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    reportError<SQLHDBC>(SQL_HANDLE_DBC, hdlDbc);
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}
// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
SQL_NTS);
// Create a table into which we can store data
printf("Creating table.\n");
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
"(CustID int, CustName varchar(50), Phone_Number char(15));",
SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);
    exit(EXIT_FAILURE);
} else {
    printf("Created table.\n");
}
// Create the prepared statement. This will insert data into the
```

```
// table we created above.
printf("Creating prepared statement\n");
ret = SQLPrepare (hdlStmt, (SQLTCHAR*)"INSERT INTO customers (CustID, "
    "CustName, Phone_Number) VALUES(?,?,?)", SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
} else {
    printf("Created prepared statement.\n");
}
// This is the data to be inserted into the database.
char custNames[][50] = { "Allen, Anna", "Brown, Bill", "Chu, Cindy",
    "Dodd, Don" };
SQLINTEGER custIDs[] = { 100, 101, 102, 103};
char phoneNums[][15] = {"1-617-555-1234", "1-781-555-1212",
    "1-508-555-4321", "1-617-555-4444"};
// Bind the data arrays to the parameters in the prepared SQL
// statement
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
    0, 0, (SQLPOINTER)custIDs, sizeof(SQLINTEGER), NULL);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
} else {
    printf("Bound CustIDs array to prepared statement\n");
}
// Bind CustNames
SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
    50, 0, (SQLPOINTER)custNames, 50, NULL);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
} else {
    printf("Bound CustNames array to prepared statement\n");
}
// Bind phoneNums
SQLBindParameter(hdlStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
    15, 0, (SQLPOINTER)phoneNums, 15, NULL);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNums array to prepared statement\n");
}
// Set up a variable to receive number of parameters processed.
SQLULEN rowsProcessed;
// Set a statement attribute to point to the variable
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &rowsProcessed, 0);
// Set up an array to hold the result of each row insert
SQLUSMALLINT rowResults[ NUM_ENTRIES ];
// Set a statement attribute to point to the array
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAM_STATUS_PTR, rowResults, 0);
// Tell the ODBC driver how many rows we have in the
// array.
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)NUM_ENTRIES, 0);
// Add multiple batches to the database. This just adds the same
// batch of data over and over again for simplicity's sake.
for (int batchLoop=1; batchLoop<=5; batchLoop++) {
    // Execute the prepared statement, loading all of the data
    // in the arrays.
}
```

```
printf("Adding Batch #%d...", batchLoop);
ret = SQLExecute(hdlStmt);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
}
// Number of rows processed is in rowsProcessed
printf("Params processed: %d\n", rowsProcessed);
printf("Results of inserting each row:\n");
int i;
for (i = 0; i<NUM_ENTRIES; i++) {
    SQLUSMALLINT result = rowResults[i];
    switch(rowResults[i]) {
        case SQL_PARAM_SUCCESS:
        case SQL_PARAM_SUCCESS_WITH_INFO:
            printf(" Row %d inserted successssfully\n", i+1);
            break;
        case SQL_PARAM_ERROR:
            printf(" Row %d was not inserted due to an error.", i+1);
            break;
        default:
            printf(" Row %d had some issue with it: %d\n", i+1, result);
    }
}
}
// Done with batches, commit the transaction
printf("Commit Transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

Running the example code produces the following output:

```
Allocated an environment handle.Set application to ODBC 3.
Allocated Database handle.
Connecting to database.
Connected to database.
Creating table.
Created table.
Creating prepared statement
Created prepared statement.
Bound CustIDs array to prepared statement
Bound CustNames array to prepared statement
```

```
Bound phoneNums array to prepared statement
Adding Batch #1...Params processed: 4
Results of inserting each row:
  Row 1 inserted successfully
  Row 2 inserted successfully
  Row 3 inserted successfully
  Row 4 inserted successfully
Adding Batch #2...Params processed: 4
Results of inserting each row:
  Row 1 inserted successfully
  Row 2 inserted successfully
  Row 3 inserted successfully
  Row 4 inserted successfully
Adding Batch #3...Params processed: 4
Results of inserting each row:
  Row 1 inserted successfully
  Row 2 inserted successfully
  Row 3 inserted successfully
  Row 4 inserted successfully
Adding Batch #4...Params processed: 4
Results of inserting each row:
  Row 1 inserted successfully
  Row 2 inserted successfully
  Row 3 inserted successfully
  Row 4 inserted successfully
Adding Batch #5...Params processed: 4
Results of inserting each row:
  Row 1 inserted successfully
  Row 2 inserted successfully
  Row 3 inserted successfully
  Row 4 inserted successfully
Commit Transaction
Free handles.
```

Error Handling During Batch Loads

When loading individual batches, you can find information on how many rows were accepted and what rows were rejected (see [Finding the Number of Accepted Rows](#) for details). Other errors, such as disk space errors, do not occur while inserting individual batches. This behavior is caused by having a single COPY statement perform the loading of multiple consecutive batches. Using the single COPY statement makes the batch load process perform much faster. It is only when the COPY statement closes that the batched data is committed and Vertica reports other types of errors.

Your bulk loading application should check for errors when the COPY statement closes. Normally, you force the COPY statement to close by calling the `SQLEndTran()` function to end the transaction. You can also force the COPY statement to close by closing the cursor using the `SQLCloseCursor()` function, or by setting the database connection's `AutoCommit` property to true before inserting the last batch in the load.

Note: The COPY statement also closes if you execute any non-insert statement. However having to deal with errors from the COPY statement in what might be an otherwise-unrelated query is not intuitive, and can lead to confusion and a harder to maintain application. You should explicitly end the COPY statement at the end of your batch load and handle any errors at that time.

Using the COPY Statement

The COPY statement lets you bulk load data from a file on stored on a database node into the Vertica database. This method is the most efficient way to load data into Vertica because the file resides on the database server. One drawback is that only a database superuser can use COPY, since it requires privilege in order to access the filesystem of the database node.

One drawback of using COPY instead of performing batch loads is that you can only get results of the load (the number of accepted and rejected rows) when the COPY statement has finished. With batch loads, you can monitor the progress as batches are inserted. The ability to monitor the progress of a load can be a useful feature if you want to stop loading if a large portion of the data is being rejected.

A primary concern when bulk loading data using COPY is deciding whether the data should be loaded directly into ROS using the DIRECT option, or by using the AUTO method (loading into WOS until it fills, then loading into ROS). You should load directly into the ROS when your transaction will load a large (more than 100MB of data or so) amount of data.

Note: The exceptions/rejections files are created on the client machine when the exceptions and rejected data modifiers are specified on the COPY command. Specify a local path and filename for these modifiers when executing a COPY query from the driver.

The following example loads data into the WOS (Write Optimized Store) until it fills, then stores additional data directly in ROS (Read Optimized Store).

```
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers "  
"FROM '/data/customers.txt' AUTO",SQL_NTS);
```

The following example loads data into the ROS (Read Optimized Store).

```
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers "  
"FROM '/data/customers.txt' DIRECT",SQL_NTS);
```

See the [COPY](#) statement in the SQL Reference Manual for more information about its syntax and use.

The following example demonstrates using the COPY command.

```
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
// Helper function to determine if an ODBC function call returned
// successfully.
bool notSuccess(SQLRETURN ret) {
    return (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO);
}
int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(notSuccess(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    if(notSuccess(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";

    // Note: User MUST be a database superuser to be able to access files on the
    // filesystem of the node.
    const char* userID = "dbadmin";
    const char* passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
        SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
        (SQLCHAR*)passwd, SQL_NTS);
    if(notSuccess(ret)) {
        printf("Could not connect to database.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Connected to database.\n");
    }

    // Disable AUTOCOMMIT
    printf("Disabling autocommit.\n");
    ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, SQL_NTS);
    if(notSuccess(ret)) {
```

```
    printf("Could not disable autocommit.\n");
    exit(EXIT_FAILURE);
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
// Create table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
    SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers"
    "(Last_Name char(50) NOT NULL, First_Name char(50),Email char(50), "
    "Phone_Number char(15));",
    SQL_NTS);

// Run the copy command to load data into ROS.
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers "
    "FROM '/data/customers.txt' DIRECT",
    SQL_NTS);
if(notSuccess(ret)) {
    printf("Data was not successfully loaded.\n");
    exit(EXIT_FAILURE);
} else {
    // Get number of rows added.
    SQLENUM numRows;
    ret=SQLRowCount(hdlStmt, &numRows);
    printf("Successfully inserted %d rows.\n", numRows);
}

// Done with batches, commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(notSuccess(ret)) {
    printf("Could not commit transaction\n");
} else {
    printf("Committed transaction\n");
}

// Clean up
printf("Free handles.\n");
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

The example prints the following when run:

```
Allocated an environment handle.
Set application to ODBC 3.
Connecting to database.
Connected to database.
Disabling autocommit.
Successfully inserted 10001 rows.
Committing transaction
Committed transaction
```

Free handles.

Streaming Data From the Client Using COPY LOCAL

The LOCAL option of the SQL [COPY](#) statement lets you stream data from a file on a client system to your Vertica database. This statement works through the ODBC driver, making the task of transferring data files from the client to the server much easier.

The LOCAL option of COPY works transparently through the ODBC driver. Just have your client application execute a statement containing a COPY LOCAL statement, and the ODBC driver will read and stream the data file from the client to the server. For example:

```
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Connect to the database
    printf("Connecting to database.\n");
}
```

```
const char *dsnName = "ExampleDB";
const char* userID = "dbadmin";
const char* passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
    SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
    (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Create table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
    SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers"
    "(Last_Name char(50) NOT NULL, First_Name char(50),Email char(50), "
    "Phone_Number char(15));",
    SQL_NTS);

// Run the copy command to load data into ROS.
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers "
    "FROM LOCAL '/home/dbadmin/customers.txt' DIRECT",
    SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Data was not successfully loaded.\n");
    exit(EXIT_FAILURE);
} else {
    // Get number of rows added.
    SQLLEN numRows;
    ret=SQLRowCount(hdlStmt, &numRows);
    printf("Successfully inserted %d rows.\n", numRows);
}

// COPY commits automatically, unless it is told not to, so
// there is no need to commit the transaction.

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}
```

This example is essentially the same as the example shown in [Using the COPY Statement](#), except it uses the COPY statement's LOCAL option to load data from the client system rather than from the filesystem of the database node.

Note: On Windows clients, the path you supply for the COPY LOCAL file is limited to 216 characters due to limitations in the Windows API.

See Also

- [COPY LOCAL](#)

Using LONG VARCHAR and LONG VARBINARY Data Types with ODBC

The ODBC drivers support the LONG VARCHAR and LONG VARBINARY data types similarly to VARCHAR and VARBINARY data types. When binding input or output parameters to a LONG VARCHAR or LONG VARBINARY column in a query, use the SQL_LONGVARCHAR and SQL_LONGVARBINARY constants to set the column's data type. For example, to bind an input parameter to a LONG VARCHAR column, you would use a statement that looks like this:

```
rc = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR,  
80000, 0, (SQLPOINTER)myLongString, sizeof(myLongString), NULL);
```

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

Programming JDBC Client Applications

The Vertica JDBC driver provides you with a standard JDBC API. If you have accessed other databases using JDBC, you should find accessing Vertica familiar. This section explains how to use the JDBC to connect your Java application to Vertica.

You must first install the JDBC client driver on all client systems that will be accessing the Vertica database. For installation instructions, see [Installing the Vertica Client Drivers](#).

For more information about JDBC:

- [Vertica Analytics Platform JDBC Documentation \(Vertica extensions\)](#)
- [An Introduction to JDBC, Part 1](#)

JDBC Feature Support

The Vertica JDBC driver complies with the JDBC 4.0 standards (although it does not implement all of the optional features in them). Your application can use the `DatabaseMetaData` class to determine if the driver supports a particular feature it wants to use. In addition, the driver implements the `Wrapper` interface, which lets your client code discover Vertica-specific extensions to the JDBC standard classes, such as `VerticaConnection` and `VerticaStatement` classes.

Some important facts to keep in mind when using the Vertica JDBC driver:

- Cursors are forward only and are not scrollable. Result sets cannot be updated.
- A connection supports executing a single statement at any time. If you want to execute multiple statements simultaneously, you must open multiple connections.
- Because Vertica does not have stored procedures, `CallableStatement` is not supported. The `DatabaseMetaData.getProcedures()` and `.getProcedureColumns()` methods return information about SQL functions (including User Defined Functions) instead of stored procedures.

Multiple SQL Statement Support

The Vertica JDBC driver can execute strings containing multiple statements. For example:

```
stmt.executeUpdate("CREATE TABLE t(a INT);INSERT INTO t VALUES(10);");
```

Only the `Statement` interface supports executing strings containing multiple SQL statements. You cannot use multiple statement strings with `PreparedStatement`. `COPY` statements that copy a file from a host file system work in a multiple statement string. However, client `COPY` statements (`COPY FROM STDIN`) do not work.

Multiple Batch Conversion to COPY Statements

The Vertica JDBC driver converts all batch inserts into Vertica `COPY` statements. If you turn off your JDBC connection's `AutoCommit` property, the JDBC driver uses a single `COPY` statement to

load data from sequential batch inserts which can improve load performance by reducing overhead. See [Batch Inserts Using JDBC Prepared Statements](#) for details.

Multiple JDBC Version Support

The Vertica JDBC driver implements both JDBC 3.0 and JDBC 4.0 compliant interfaces. The interface that the driver returns to your application depends on the JVM version on which it is running. If your application is running on a 5.0 JVM, the driver supplies your application with JDBC 3.0 classes. If your application is running on a 6.0 or later JVM, the driver supplies it with JDBC 4.0 classes.

Multiple Active Result Sets (MARS)

The Vertica JDBC driver supports [Multiple Active Result Sets \(MARS\)](#). MARS allows the execution of multiple queries on a single connection. While [ResultSet](#) sends the results of a query directly to the client, MARS stores the results first on the server. Once query execution has finished and all of the results have been stored, you can make a retrieval request to the server to have rows returned to the client.

Creating and Configuring a Connection

Before your Java application can interact with Vertica, it must create a connection. Connecting to Vertica using JDBC is similar to connecting to most other databases.

Importing SQL Packages

Before creating a connection, you must import the Java SQL packages. A simple way to do so is to import the entire package using a wildcard:

```
import java.sql.*;
```

You may also want to import the `Properties` class. You can use an instance of this class to pass connection properties when instantiating a connection, rather than encoding everything within the connection string:

```
import java.util.Properties;
```

If your application needs to run in a Java 5 JVM, it uses the older JDBC 3.0-compliant driver. This driver requires that you manually load the Vertica JDBC driver using the `Class.forName()` method:

```
// Only required for old JDBC 3.0 driver
try {
    Class.forName("com.vertica.jdbc.Driver");
} catch (ClassNotFoundException e) {
    // Could not find the driver class. Likely an issue
    // with finding the .jar file.
    System.err.println("Could not find the JDBC driver class.");
    e.printStackTrace();
    return; // Exit. Cannot do anything further.
}
```

Your application may run in a Java 6 or later JVM. If so, then the JVM automatically loads the Vertica JDBC 4.0-compatible driver without requiring the call to `Class.forName`. However, making this call does not adversely affect the process. Thus, if you want your application to be compatible with both Java 5 and Java 6 (or later) JVMs, it can still call `Class.forName`.

Opening the Connection

With SQL packages imported, you are ready to create your connection by calling the `DriverManager.getConnection()` method. You supply this method with at least the following information:

- The IP address or host name of a node in the database cluster:

You can provide an IPv4 address, IPv6 address, or host name.

In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the `PreferredAddressFamily` option to force the connection to use either IPv4 or IPv6.

- Port number for the database
- Name of the database
- Username of a database user account
- Password of the user (if the user has a password)

The first three parameters are always supplied as part of the *connection string*, a URL that tells the JDBC driver where to find the database. The format of the connection string is:

```
"jdbc:vertica://VerticaHost:portNumber/databaseName"
```

The first portion of the connection string selects the Vertica JDBC driver, followed by the location of the database.

You can provide the last two parameters, username and password, to the JDBC driver, in one of three ways:

- As part of the connection string. The parameters are encoded similarly to URL parameters:

```
"jdbc:vertica://VerticaHost:portNumber/databaseName?user=username&password=password"
```

- As separate parameters to `DriverManager.getConnection()`:

```
Connection conn = DriverManager.getConnection(  
    "jdbc:vertica://VerticaHost:portNumber/databaseName",  
    "username", "password");
```

- In a `Properties` object:

```
Properties myProp = new Properties();  
myProp.put("user", "username");  
myProp.put("password", "password");  
Connection conn = DriverManager.getConnection(  
    "jdbc:vertica://VerticaHost:portNumber/databaseName", myProp);
```

Of these three methods, the `Properties` object is the most flexible because it makes passing additional connection properties to the `getConnection()` method easy. See [Connection](#)

[Properties](#) and [Setting and Getting Connection Property Values](#) for more information about the additional connection properties.

If there is any problem establishing a connection to the database, the `getConnection()` method throws a `SQLException` on one of its subclasses. . To prevent an exception, enclose the method within a try-catch block, as shown in the following complete example of establishing a connection.

```
import java.sql.*;
import java.util.Properties;

public class VerySimpleVerticaJDBCExample {
    public static void main(String[] args) {
        /*
         * If your client needs to run under a Java 5 JVM, It will use the older
         * JDBC 3.0-compliant driver, which requires you manually load the
         * driver using Class.forName
         */
        /*
         * try { Class.forName("com.vertica.jdbc.Driver"); } catch
         * (ClassNotFoundException e) { // Could not find the driver class.
         * Likely an issue // with finding the .jar file.
         * System.err.println("Could not find the JDBC driver class.");
         * e.printStackTrace(); return; // Bail out. We cannot do anything
         * further. }
         */
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "vertica");
        myProp.put("loginTimeout", "35");
        myProp.put("binaryBatchInsert", "true");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://V_vmart_node0001.example.com:5433/vmart", myProp);
            System.out.println("Connected!");
            conn.close();
        } catch (SQLTransientConnectionException connException) {
            // There was a potentially temporary network error
            // Could automatically retry a number of times here, but
            // instead just report error and exit.
            System.out.print("Network connection issue: ");
            System.out.print(connException.getMessage());
            System.out.println(" Try again later!");
            return;
        } catch (SQLInvalidAuthorizationSpecException authException) {
            // Either the username or password was wrong
            System.out.print("Could not log into database: ");
            System.out.print(authException.getMessage());
            System.out.println(" Check the login credentials and try again.");
            return;
        } catch (SQLException e) {
            // Catch-all for other exceptions
            e.printStackTrace();
        }
    }
}
```

Usage Considerations

- When you disconnect a user session, any uncommitted transactions are automatically rolled back.
- If your database is not compliant with your Vertica license terms, Vertica issues a `SQLWarning` when you establish the connection to the database. You can retrieve this warning using the `Connection.getWarnings()` method. See [Managing Licenses](#) in the Administrator's Guide for more information about complying with your license terms.

JDBC Connection Properties

You use connection properties to configure the connection between your JDBC client application and your Vertica database. The properties provide the basic information about the connections, such as the server name and port number to use to connect to your database. They also let you tune the performance of your connection and enable logging.

You can set a connection property in any of three ways:

- Include the property name and value as part of the connection string you pass to the `DriverManager.getConnection()` method.
- Set the properties in a `Properties` object, and then pass it to the `DriverManager.getConnection()` method.
- Use the `VerticaConnection.setProperty()` method. With this approach, you can change only those connection properties that remain changeable after the connection has been established.

In addition, some of the standard JDBC connection properties have getters and setters on the `Connection` interface (such as `Connection.setAutoCommit()`).

The following tables list the properties supported by the Vertica JDBC driver, and explain which are required in order for the connection to be established.

Connection Properties

The properties in the following table can only be set before you open the connection to the database. Two of them are required for every connection.

Property	Description
ConnSettings	<p>A string containing SQL statements that the JDBC driver automatically runs after it connects to the database. You can use this property to set the locale or schema search path, or perform other configuration that the connection requires.</p> <p>Required?: No</p> <p>Default Value: none</p>
DisableCopyLocal	<p>When set to true, disables file-based COPY LOCAL operations, including copying data from local files and using local files to store data and exceptions. You can use this property to prevent users from writing to and copying from files on a Vertica host, including an MC host.</p> <p>Required?: No</p> <p>Default Value: false</p>
Label	<p>Sets a label for the connection on the server. This value appears in the client_label column of the V_MONITOR.SESIONS system table.</p> <p>Required?: No</p> <p>Default Value: jdbc-driver_version-random_number</p>
LoginTimeout	<p>The number of seconds Vertica waits for a connection to be established to the database before throwing a SQLException.</p> <p>Required?: No</p> <p>Default Value: 0 (no TCP timeout)</p>
SSL	<p>When set to true, use SSL to encrypt the connection to the server. Vertica must be configured to handle SSL connections before you can establish an SSL-encrypted connection to it. See TLS/SSL Server Authentication in the Administrator's Guide.</p> <p>Required?: No</p> <p>Default Value: false</p>
Password	<p>The password to use to log into the database.</p> <p>Required?: Yes</p> <p>Default Value: none</p>

Property	Description
User	<p>The database user name to use to connect to the database.</p> <p>Required?: Yes</p> <p>Default Value: none</p>
ConnectionLoadBalance	<p>A Boolean value indicating whether the client is willing to have its connection redirected to another host in the Vertica database. This setting has an effect only if the server has also enabled connection load balancing. See About Native Connection Load Balancing in the Administrator's Guide for more information about native connection load balancing.</p> <p>Required?: No</p> <p>Default Value: false</p>
BackupServerNode	<p>A string containing the host name or IP address of one or more hosts in the database. If the connection to the host specified in the connection string times out, the client attempts to connect to any host named in this string. The host name or IP address can also include a colon followed by the port number for the database. If no port number is specified, the client uses the standard port number (5433) . Separate multiple host name or IP address entries with commas.</p> <p>Required?: No</p> <p>Default Value: none</p>
PreferredAddressFamily	<p>The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:</p> <ul style="list-style-type: none">• ipv4—Connect to the server using IPv4.• ipv6—Connect to the server using IPv6.• none—Use the IP address provided by the DNS server. <p>Required?: No</p> <p>Default Value: none</p>

General Properties

The following properties can be set after the connection is established. None of these properties are required.

Property	Description
AutoCommit	<p>Controls whether the connection automatically commits transactions. Set this parameter to false to prevent the connection from automatically committing its transactions. You often want to do this when you are bulk loading multiple batches of data and you want the ability to roll back all of the loads if an error occurs.</p> <p>Previous Property NameNote: This property was called defaultAutoCommit in previous versions of the Vertica JDBC driver.</p> <p>Set After Connection: <code>Connection.setAutoCommit()</code></p> <p>Default Value: true</p>
DirectBatchInsert	<p>Determines whether a batch insert stored data directly into ROS (true) or using AUTO mode (false).</p> <p>When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</p> <p>Set After Connection: <code>VerticaConnection.setProperty()</code></p> <p>Default Value: false</p>
MultipleActiveResultSets	<p>Allows more than one active result set on a single connection via MultipleActiveResultSets (MARS).</p> <p>If both MultipleActiveResultSets and ResultBufferSize are turned on, MultipleActiveResultSets takes precedence. The connection does not provide an error, however ResultBufferSize is ignored.</p> <p>Set After Connection: <code>VerticaConnection.setProperty()</code></p> <p>Default Value: false</p>

Property	Description
ReadOnly	<p>When set to true, makes the data connection read-only. Any queries attempting to update the database using a read-only connection cause a <code>SQLException</code>.</p> <p>Set After Connection: <code>Connection.setReadOnly()</code></p> <p>Default Value: false</p>
ResultBufferSize	<p>Sets the size of the buffer the Vertica JDBC driver uses to temporarily store result sets. A value of 0 means ResultBufferSize is turned off.</p> <p>Note: This property was named <code>maxLRSMemory</code> in previous versions of the Vertica JDBC driver.</p> <p>Set After Connection: <code>VerticaConnection.setProperty()</code></p> <p>Default Value: 8912 (8KB)</p>
SearchPath	<p>Sets the schema search path for the connection. This value is a string containing a comma-separated list of schema names. See Setting Search Paths in the Administrator's Guide for more information on the schema search path.</p> <p>Set After Connection: <code>VerticaConnection.setProperty()</code></p> <p>Default Value: "\$user", public, v_catalog, v_monitor, v_internal</p>
ThreePartNaming	<p>A Boolean value that controls how DatabaseMetaData reports the catalog name. When set to true, the database name is returned as the catalog name in the database metadata. When set to false, NULL is returned as the catalog name.</p> <p>Enable this option if your client software is set up to get the catalog name from the database metadata for use in a three-part name reference.</p> <p>Set After Connection: <code>VerticaConnection.setProperty()</code></p> <p>Default Value: true</p>
TransactionIsolation	<p>Sets the isolation level of the transactions that use the</p>

Property	Description
	<p>connection. See Changing the Transaction Isolation Level for details.</p> <p>Note: In previous versions of the Vertica JDBC driver, this property was only available using a getter and setter on the <code>PGConnection</code> object. You can now set it in the same way as other connection properties.</p> <p>Set After Connection: <code>Connection.setTransactionIsolation()</code></p> <p>Default Value: TRANSACTION_READ_COMMITTED</p>

Logging Properties

The properties that control client logging must be set before the connection is opened. None of these properties are required, and none can be changed after the `Connection` object has been instantiated.

Property	Description
LogLevel	<p>Sets the type of information logged by the JDBC driver. The value is set to one of the following values:</p> <ul style="list-style-type: none">• "DEBUG"• "ERROR"• "TRACE"• "WARNING"• "INFO"• "OFF" <p>Default Value: "OFF"</p>
LogNameSpace	<p>Restricts logging to just messages generated by a specific packages. Valid values are:</p> <ul style="list-style-type: none">• <code>com.vertica</code> — All messages generated by the JDBC driver• <code>com.vertica.jdbc</code> — All messages generated by the top-level

Property	Description
	<p>JDBC API</p> <ul style="list-style-type: none"> • <code>com.vertica.jdbc.kv</code> — All messages generated by the JDBC KV API) • <code>com.vertica.jdbc.core</code> — Connection and statement settings • <code>com.vertica.jdbc.io</code> — Client/server protocol messages • <code>com.vertica.jdbc.util</code> — Miscellaneous utilities • <code>com.vertica.jdbc.dataengine</code> — Query execution and result set iteration • <code>com.vertica.dataengine</code> — Query execution and result set iteration <p>Default Value: none</p>
LogPath	<p>Sets the path where the log file is written.</p> <p>Default Value: The current working directory</p>

Kerberos Connection Parameters

Use the following parameters to set the service and host name principals for client authentication using Kerberos.

Parameters	Description
JAASConfigName	<p>Provides the name of the JAAS configuration that contains the JAAS Krb5LoginModule and its settings</p> <p>Default Value: verticajdbc</p>
KerberosServiceName	<p>Provides the service name portion of the Vertica Kerberos principal, for example: <code>vertica/host@EXAMPLE.COM</code></p> <p>Default Value: vertica</p>
KerberosHostname	<p>Provides the instance or host name portion of the Vertica Kerberos principal, for example: <code>vertica/host@EXAMPLE.COM</code></p> <p>Default Value: Value specified in the servername connection string property</p>

Routable Connection API Connection Parameters

Use the following parameters to set properties to enable and configure the connection for Routable Connection lookups.

Parameters	Description
EnableRoutableQueries	Enables Routable Connection lookup. See Routing JDBC Queries Directly to a Single Node Default Value: false
FailOnMultiNodePlans	If the query plan requires more than one node, then the query fails. Only applicable when EnableRoutableQueries = true. Default Value: true
MetadataCacheLifetime	The time in seconds to keep projection metadata. Only applicable when EnableRoutableQueries = true. Default Value:
MaxPooledConnections	Cluster-wide maximum number of connections to keep in the VerticaRoutableConnection's internal pool. Only applicable when EnableRoutableQueries = true. Default Value: 20
MaxPooledConnections PerNode	Per-node maximum number of connections to keep in the VerticaRoutableConnection's internal pool. Only applicable when EnableRoutableQueries = true. Default Value: 5

Note: You can also use `VerticaConnection.setProperty()` method to set properties that have standard JDBC Connection setters, such as `AutoCommit`.

For information about manipulating these attributes, see [Setting and Getting Connection Property Values](#).

Setting and Getting Connection Property Values

You can set a connection property in any of three ways:

- Include the property name and value as part of the connection string you pass to the `DriverManager.getConnection()` method.
- Set the properties in a `Properties` object, and then pass it to the `DriverManager.getConnection()` method.
- Use the `VerticaConnection.setProperty()` method. With this approach, you can change only those connection properties that remain changeable after the connection has been established.

In addition, some of the standard JDBC connection properties have getters and setters on the `Connection` interface (such as `Connection.setAutoCommit()`).

Setting Properties When Connecting

There are two ways you can set connection properties when creating a connection to Vertica:

- In the connection string, using the same URL parameter format that you can use to set the username and password. The following example sets the SSL connection parameter to true:

```
"jdbc:vertica://VerticaHost:5433/db?user=UserName&password=Password&ssl=true"
```

Setting a host name using the `setProperty()` method overrides the host name set in a connection string as seen above. If this occurs, Vertica may not be able to connect to a host. For example, using the connection string above, the following overrides the `VerticaHost` name:

```
Properties props = new Properties();
props.setProperty("dataSource", dataSourceURL);
props.setProperty("database", database);
props.setProperty("user", user);
props.setProperty("password", password);
ps.setProperty("jdbcDriver", jdbcDriver);
props.setProperty("hostName", "NonVertica_host");
```

However, if a new connection or override connection is needed, you may enter a valid host name in the hostname properties object.

The `NonVertica_host` hostname overrides `VerticaHost` name in the connection string. To avoid this issue comment out the `props.setProperty("hostName", "NonVertica_host");` line:

```
//props.setProperty("hostName", "NonVertica_host");
```

- In a `Properties` object that you pass to the `getConnection()` call. You will need to import the `java.util.Properties` class in order to instantiate a `Properties` object. Then you use the `put()` method to add the property name and value to the object:

```
Properties myProp = new Properties();
myProp.put("user", "ExampleUser");
myProp.put("password", "password123");
myProp.put("LoginTimeout", "35");
Connection conn;
try {
    conn = DriverManager.getConnection(
        "jdbc:vertica://VerticaHost:/ExampleDB", myProp);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Note: The data type of all of the values you set in the `Properties` object are strings, regardless of the property value's data type.

Getting and Setting Properties After Connecting

The `VerticaConnection.getProperty()` method lets you get the value of some connection properties. You can use `VerticaConnection.setProperty()` method to change the value for properties that can be set after the database connection has been established. Since these methods are Vertica-specific, to use them you must cast your `Connection` object to the `VerticaConnection` interface. To cast to `VerticaConnection`, you must either import it into your client application or use a fully-qualified reference (`com.vertica.jdbc.VerticaConnection`). The following example demonstrates getting and setting the value of the `DirectBatchInsert` property.

```
import java.sql.*;
import java.util.Properties;
import com.vertica.jdbc.*;

public class SetConnectionProperties {
    public static void main(String[] args) {
        // Note: If your application needs to run under Java 5, you need to
        // load the JDBC driver using Class.forName() here.
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        // Set DirectBatchInsert to true initially
        myProp.put("DirectBatchInsert", "true");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",
                myProp);
            // Show state of the DirectBatchInsert property. This was set at the
```

```
// time the connection was created.
System.out.println("DirectBatchInsert state: "
    + ((VerticaConnection) conn).getProperty(
        "DirectBatchInsert"));

// Change it and show it again
((VerticaConnection) conn).setProperty("DirectBatchInsert", false);
System.out.println("DirectBatchInsert state is now: " +
    ((VerticaConnection) conn).getProperty(
        "DirectBatchInsert"));
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

When run, the example prints the following on the standard output:

```
DirectBatchInsert state: true
DirectBatchInsert state is now: false
```

Setting and Returning a Client Connection Label

The JDBC Client has a method to set and return the client connection label: `getClientInfo()` and `setClientInfo()`. You can use these methods with the SQL Functions [GET_CLIENT_LABEL](#) and [SET_CLIENT_LABEL](#).

When you use these two methods, make sure you pass the string value `APPLICATIONNAME` to both the setter and getter methods.

Use `setClientInfo()` to create a client label, and use `getClientInfo()` to return the client label:

```
import java.sql.*;
import java.util.Properties;

public class ClientLabelJDBC {

    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "");
        myProp.put("loginTimeout", "35");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://docc05.verticacorp.com:5433/doccdb", myProp);
            System.out.println("Connected!");
            conn.setClientInfo("APPLICATIONNAME", "JDBC Client - Data Load");
            System.out.println("New Conn label: " + conn.getClientInfo("APPLICATIONNAME"));
            conn.close();
        } catch (SQLTransientConnectionException connException) {
            // There was a potentially temporary network error
        }
    }
}
```

```
// Could automatically retry a number of times here, but
// instead just report error and exit.
System.out.print("Network connection issue: ");
System.out.print(connException.getMessage());
System.out.println(" Try again later!");
return;
} catch (SQLException e) {
    // Either the username or password was wrong
    System.out.print("Could not log into database: ");
    System.out.print(e.getMessage());
    System.out.println(" Check the login credentials and try again.");
    return;
} catch (SQLException e) {
    // Catch-all for other exceptions
    e.printStackTrace();
}
}
```

When you run this method, it prints the following result to the standard output:

```
Connected!
New Conn Label: JDBC Client - Data Load
```

Setting the Locale for JDBC Sessions

You set the locale for a connection while opening it by including a `SET LOCALE` statement in the `ConnSettings` property, or by executing a [SET LOCALE](#) statement at any time after opening the connection. Changing the locale of a `Connection` object affects all of the `Statement` objects you instantiated using it.

You can get the locale by executing a [SHOW LOCALE](#) query. The following example demonstrates setting the locale using `ConnSettings` and executing a statement, as well as getting the locale:

```
import java.sql.*;
import java.util.Properties;

public class GetAndSetLocale {
    public static void main(String[] args) {

        // If running under a Java 5 JVM, you need to load the JDBC driver
        // using Class.forName here

        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");

        // Set Locale to true en_GB on connection. After the connection
        // is established, the JDBC driver runs the statements in the
        // ConnSettings property.
        myProp.put("ConnSettings", "SET LOCALE TO en_GB");
    }
}
```

```
Connection conn;
try {
    conn = DriverManager.getConnection(
        "jdbc:vertica://VerticaHost:5433/ExampleDB",
        myProp);

    // Execute a query to get the locale. The results should
    // show "en_GB" as the locale, since it was set by the
    // conn settings property.
    Statement stmt = conn.createStatement();
    ResultSet rs = null;
    rs = stmt.executeQuery("SHOW LOCALE");
    System.out.print("Query reports that Locale is set to: ");
    while (rs.next()) {
        System.out.println(rs.getString(2).trim());
    }

    // Now execute a query to set locale.
    stmt.execute("SET LOCALE TO en_US");

    // Run query again to get locale.
    rs = stmt.executeQuery("SHOW LOCALE");
    System.out.print("Query now reports that Locale is set to: ");
    while (rs.next()) {
        System.out.println(rs.getString(2).trim());
    }
    // Clean up
    conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

Running the above example displays the following on the system console:

```
Query reports that Locale is set to: en_GB (LEN)
Query now reports that Locale is set to: en_US (LEN)
```

Notes:

- JDBC applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. Failing to convert the data can result in errors or the data being stored incorrectly.
- The JDBC driver converts UTF-16 data to UTF-8 when passing to the Vertica server and converts data sent by Vertica server from UTF-8 to UTF-16 .

Changing the Transaction Isolation Level

Changing the transaction isolation level lets you choose how transactions prevent interference from other transactions. By default, the JDBC driver matches the transaction isolation level of the Vertica server. The Vertica default transaction isolation level is `READ_COMMITTED`, which means any changes made by a transaction cannot be read by any other transaction until after they are committed. This prevents a transaction from reading data inserted by another transaction that is later rolled back.

Vertica also supports the `SERIALIZABLE` transaction isolation level. This level locks tables to prevent queries from having the results of their `WHERE` clauses changed by other transactions. Locking tables can have a performance impact, since only one transaction is able to access the table at a time.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the `SERIALIZABLE` isolation level to ensure consistency.

You can change the transaction isolation level connection property after the connection has been established using the `Connection` object's setter (`setTransactionIsolation()`) and getter (`getTransactionIsolation()`). The value for transaction isolation property is an integer. The `Connection` interface defines constants to help you set the value in a more intuitive manner:

Constant	Value
<code>Connection.TRANSACTION_READ_COMMITTED</code>	2
<code>Connection.TRANSACTION_SERIALIZABLE</code>	8

Note: The `Connection` interface also defines several other transaction isolation constants (`READ_UNCOMMITTED` and `REPEATABLE_READ`). Since Vertica does not support these isolation levels, they are converted to `READ_COMMITTED` and `SERIALIZABLE`, respectively.

The following example demonstrates setting the transaction isolation level to `SERIALIZABLE`.

```
import java.sql.*;
import java.util.Properties;

public class SetTransactionIsolation {
```

```
public static void main(String[] args) {
    Properties myProp = new Properties();
    myProp.put("user", "ExampleUser");
    myProp.put("password", "password123");
    Connection conn;
    try {
        conn = DriverManager.getConnection(
            "jdbc:vertica://VerticaHost:5433/ExampleDB",
            myProp);
        // Get default transaction isolation
        System.out.println("Transaction Isolation Level: "
            + conn.getTransactionIsolation());
        // Set transaction isolation to SERIALIZABLE
        conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
        // Get the transaction isolation again
        System.out.println("Transaction Isolation Level: "
            + conn.getTransactionIsolation());
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Running the example results in the following being printed out to the console:

```
Transaction Isolation Level: 2Transaction Isolation Level: 8
```

Using a Pooling Data Source

A pooling data source uses a collection of persistent connections in order to reduce the overhead of repeatedly opening network connections between the client and server. Opening a new connection for each request is costly for both the server and the client than keeping a small pool of connections open constantly, ready to be used by new requests. When a request comes in, one of the pre-existing connections in the pool is assigned to it. Only if there are no free connections in the pool is a new connection created. Once the request is complete, the connection returns to the pool and waits to service another request.

The Vertica JDBC driver supports connection pooling as defined in the JDBC 4.0 standard. If you are using a J2EE-based application server in conjunction with Vertica, it should already have a built-in data pooling feature. All that is required is that the application server work with the `PooledConnection` interface implemented by Vertica's JDBC driver. An application server's pooling feature is usually well-tuned for the works loads that the server is designed to handle. See your application server's documentation for details on how to work with pooled connections. Normally, using pooled connections should be transparent in your code—you will just open connections and the application server will worry about the details of pooling them.

If you are not using an application server, or your application server does not offer connection pooling that is compatible with Vertica, you can use a third-party pooling library, such as the open-source c3p0 or DBCP libraries, to implement connection pooling.

Note: The Vertica Analytics Platform client driver's native connection load balancing feature works with third-party connection pooling supplied by application servers and third-party pooling libraries. See [Enabling Native Connection Load Balancing in JDBC](#) for more information.

Enabling Native Connection Load Balancing in JDBC

Native connection load balancing helps spread the overhead caused by client connections on the hosts in the Vertica database. Both the server and the client must enable native connection load balancing in order for it to have an effect. If both have enabled it, then when the client initially connects to a host in the database, the host picks a host to handle the client connection from a list of the currently up hosts in the database, and informs the client which host it has chosen.

If the initially-contacted host did not choose itself to handle the connection, the client disconnects, then opens a second connection to the host selected by the first host. The connection process to this second host proceeds as usual—if SSL is enabled, then SSL negotiations begin, otherwise the client begins the authentication process. See [About Native Connection Load Balancing](#) in the Administrator's Guide for details.

To enable native load balancing on your client, set the `ConnectionLoadBalance` connection parameter to `true`. The following example demonstrates connecting to the database several times with native connection load balancing enabled, and fetching the name of the node handling the connection from the `V_MONITOR.CURRENT_SESSION` system table.

```
import java.sql.*;
import java.util.Properties;
import java.sql.*;
import java.util.Properties;

public class JDBCLoadingBalanceExample {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "example_password123");
        myProp.put("loginTimeout", "35");
        myProp.put("ConnectionLoadBalance", 1);
        Connection conn;
        for (int x=1; x <= 4; x++) {
            try {
                System.out.print("Connect attempt #" + x + "...");
                conn = DriverManager.getConnection(
                    "jdbc:vertica://node01.example.com:5433/vmart", myProp);
```

```
Statement stmt = conn.createStatement();
// Set the load balance policy to round robin before testing the database's load balancing.
stmt.execute("SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');");
// Query system to table to see what node we are connected to. Assume a single row
// in response set.
ResultSet rs = stmt.executeQuery("SELECT node_name FROM v_monitor.current_session;");
rs.next();
System.out.println("Connected to node " + rs.getString(1).trim());
conn.close();
} catch (SQLTransientConnectionException connException) {
// There was a potentially temporary network error
// Could automatically retry a number of times here, but
// instead just report error and exit.
System.out.print("Network connection issue: ");
System.out.print(connException.getMessage());
System.out.println(" Try again later!");
return;
} catch (SQLInvalidAuthorizationSpecException authException) {
// Either the username or password was wrong
System.out.print("Could not log into database: ");
System.out.print(authException.getMessage());
System.out.println(" Check the login credentials and try again.");
return;
} catch (SQLException e) {
// Catch-all for other exceptions
e.printStackTrace();
}
}
}
```

Running the above example produces the following output:

```
Connect attempt #1...Connected to node v_vmart_node0002
Connect attempt #2...Connected to node v_vmart_node0003
Connect attempt #3...Connected to node v_vmart_node0001
Connect attempt #4...Connected to node v_vmart_node0002
```

JDBC Connection Failover

If a client application attempts to connect to a host in the Vertica Analytics Platform cluster that is down, the connection attempt fails when using the default connection configuration. This failure usually returns an error to the user. The user must either wait until the host recovers and retry the connection or manually edit the connection settings to choose another host.

Due to Vertica Analytics Platform's distributed architecture, you usually do not care which database host handles a client application's connection. You can use the client driver's connection failover feature to prevent the user from getting connection errors when the host specified in the connection settings is unreachable. It gives you two ways to let the client driver

automatically attempt to connect to a different host if the one specified in the connection parameters is unreachable:

- Configure your DNS server to return multiple IP addresses for a host name. When you use this host name in the connection settings, the client attempts to connect to the first IP address from the DNS lookup. If the host at that IP address is unreachable, the client tries to connect to the second IP, and so on until it either manages to connect to a host or it runs out of IP addresses.
- Supply a list of backup hosts for the client driver to try if the primary host you specify in the connection parameters is unreachable.

For both methods, the process of failover is transparent to the client application (other than specifying the list of backup hosts, if you choose to use the list method of failover). If the primary host is unreachable, the client driver automatically tries to connect to other hosts.

Failover only applies to the initial establishment of the client connection. If the connection breaks, the driver does not automatically try to reconnect to another host in the database.

Choosing a Failover Method

You usually choose to use one of the two failover methods. However, they do work together. If your DNS server returns multiple IP addresses and you supply a list of backup hosts, the client first tries all of the IPs returned by the DNS server, then the hosts in the backup list.

Note: If a host name in the backup host list resolves to multiple IP addresses, the client does not try all of them. It just tries the first IP address in the list.

The DNS method of failover centralizes the configuration client failover. As you add new nodes to your Vertica Analytics Platform cluster, you can choose to add them to the failover list by editing the DNS server settings. All client systems that use the DNS server to connect to Vertica Analytics Platform automatically use connection failover without having to change any settings. However, this method does require administrative access to the DNS server that all clients use to connect to the Vertica Analytics Platform cluster. This may not be possible in your organization.

Using the backup server list is easier than editing the DNS server settings. However, it decentralizes the failover feature. You may need to update the application settings on each client system if you make changes to your Vertica Analytics Platform cluster.

Using DNS Failover

To use DNS failover, you need to change your DNS server's settings to map a single host name to multiple IP addresses of hosts in your Vertica Analytics Platform cluster. You then have all client applications use this host name to connect to Vertica Analytics Platform.

You can choose to have your DNS server return as many IP addresses for the host name as you want. In smaller clusters, you may choose to have it return the IP addresses of all of the hosts in your cluster. However, for larger clusters, you should consider choosing a subset of the hosts to return. Otherwise there can be a long delay as the client driver tries unsuccessfully to connect to each host in a database that is down.

Using the Backup Host List

To enable backup list-based connection failover, your client application has to specify at least one IP address or host name of a host in the `BackupServerNode` parameter. The host name or IP can optionally be followed by a colon and a port number. If not supplied, the driver defaults to the standard Vertica port number (5433). To list multiple hosts, separate them by a comma.

The following example demonstrates setting the `BackupServerNode` connection parameter to specify additional hosts for the connection attempt. The connection string intentionally has a non-existent node, so that the initial connection fails. The client driver has to resort to trying the backup hosts to establish a connection to Vertica.

```
import java.sql.*;
import java.util.Properties;

public class ConnectionFailoverExample {
    public static void main(String[] args) {
        // Assume using JDBC 4.0 driver on JVM 6+. No driver loading needed.
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "vertica");
        // Set two backup hosts to be used if connecting to the first host
        // fails. All of these hosts will be tried in order until the connection
        // succeeds or all of the connections fail.
        myProp.put("BackupServerNode", "VerticaHost02,VerticaHost03");
        Connection conn;
        try {
            // The connection string is set to try to connect to a known
            // bad host (in this case, a host that never existed).
            conn = DriverManager.getConnection(
                "jdbc:vertica://BadVerticaHost:5433/vmart", myProp);
            System.out.println("Connected!");
            // Query system to table to see what node we are connected to.
            // Assume a single row in response set.
            Statement stmt = conn.createStatement();
```

```
ResultSet rs = stmt.executeQuery(
    "SELECT node_name FROM v_monitor.current_session;");
rs.next();
System.out.println("Connected to node " + rs.getString(1).trim());
// Done with connection.
conn.close();
} catch (SQLException e) {
    // Catch-all for other exceptions
    e.printStackTrace();
}
}
```

When run, the example outputs output similar to the following on the system console:

```
Connected!
Connected to node v_vmart_node0002
```

Notice that the connection was made to the first node in the backup list (node 2).

Notes

- When native connection load balancing is enabled, the additional servers specified in the BackupServerNode connection parameter are only used for the initial connection to a Vertica host. If host redirects the client to another host in the database cluster to handle its connection request, the second connection does not use the backup node list. This is rarely an issue, since native connection load balancing is aware of which nodes are currently up in the database. See [Enabling Native Connection Load Balancing in JDBC](#) for more information.

JDBC Data Types

The JDBC driver transparently converts most Vertica data types to the appropriate Java data type. In a few cases, an Vertica data type cannot be directly translated to a Java data type; these exceptions are explained in this section.

Numeric Data Alias Conversion

The Vertica server supports data type aliases for integer, float and numeric types. The JDBC driver reports these as its basic data types (BIGINT, DOUBLE PRECISION, and NUMERIC), as follows:

Vertica Server Types and Aliases	Vertica JDBC Type
INTEGER INT INT8 BIGINT SMALLINT TINYINT	BIGINT
DOUBLE PRECISION FLOAT5 FLOAT8 REAL	DOUBLE PRECISION
DECIMAL NUMERIC NUMBER MONEY	NUMERIC

If a client application retrieves the values into smaller data types, Vertica JDBC driver does not check for overflows. The following example demonstrates the results of this overflow.

```
import java.sql.*;  
import java.util.Properties;
```

```
public class JDBCDataTypes {
    public static void main(String[] args) {
        // If running under a Java 5 JVM, use you need to load the JDBC driver
        // using Class.forName here

        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/VMart",
                myProp);
            Statement statement = conn.createStatement();
            // Create a table that will hold a row of different types of
            // numeric data.
            statement.executeUpdate(
                "DROP TABLE IF EXISTS test_all_types cascade");
            statement.executeUpdate("CREATE TABLE test_all_types ("
                + "c0 INTEGER, c1 TINYINT, c2 DECIMAL, "
                + "c3 MONEY, c4 DOUBLE PRECISION, c5 REAL)");
            // Add a row of values to it.
            statement.executeUpdate("INSERT INTO test_all_types VALUES("
                + "11111111111111, 444, 555555555555.5555, "
                + "77777777.77, 888888888888888888.88, "
                + "10101010.101010101010)");
            // Query the new table to get the row back as a result set.
            ResultSet rs = statement
                .executeQuery("SELECT * FROM test_all_types");
            // Get the metadata about the row, including its data type.
            ResultSetMetaData md = rs.getMetaData();
            // Loop should only run once...
            while (rs.next()) {
                // Print out the data type used to defined the column, followed
                // by the values retrieved using several different retrieval
                // methods.

                String[] vertTypes = new String[] {"INTEGER", "TINYINT",
                    "DECIMAL", "MONEY", "DOUBLE PRECISION", "REAL"};

                for (int x=1; x<7; x++) {
                    System.out.println("\n\nColumn " + x + " (" + vertTypes[x-1]
                        + ")");
                    System.out.println("\tgetColumnType()\t\t"
                        + md.getColumnType(x));
                    System.out.println("\tgetColumnTypeName()\t\t"
                        + md.getColumnTypeName(x));
                    System.out.println("\tgetShort()\t\t"
                        + rs.getShort(x));
                    System.out.println("\tgetLong()\t\t" + rs.getLong(x));
                    System.out.println("\tgetInt()\t\t" + rs.getInt(x));
                    System.out.println("\tgetBytes()\t\t" + rs.getBytes(x));
                }
            }
            rs.close();
            statement.executeUpdate("drop table test_all_types cascade");
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}  
}
```

The above example prints the following on the console when run:

```
Column 1 (INTEGER)  
  getColumnType()      -5  
  getColumnTypeName()  BIGINT  
  getShort()           455  
  getLong()            111111111111  
  getInt()             -558038585  
  getByte()            -57  
Column 2 (TINYINT)  
  getColumnType()      -5  
  getColumnTypeName()  BIGINT  
  getShort()           444  
  getLong()            444  
  getInt()             444  
  getByte()            -68  
Column 3 (DECIMAL)  
  getColumnType()      2  
  getColumnTypeName()  NUMERIC  
  getShort()           -1  
  getLong()            55555555555  
  getInt()             2147483647  
  getByte()            -1  
Column 4 (MONEY)  
  getColumnType()      2  
  getColumnTypeName()  NUMERIC  
  getShort()           -13455  
  getLong()            77777777  
  getInt()             77777777  
  getByte()            113  
Column 5 (DOUBLE PRECISION)  
  getColumnType()      8  
  getColumnTypeName()  DOUBLE PRECISION  
  getShort()           -1  
  getLong()            8888888888888900  
  getInt()             2147483647  
  getByte()            -1  
Column 6 (REAL)  
  getColumnType()      8  
  getColumnTypeName()  DOUBLE PRECISION  
  getShort()           8466  
  getLong()            10101010  
  getInt()             10101010  
  getByte()            18
```

Using Intervals with JDBC

The JDBC standard does not contain a data type for intervals (the duration between two points in time). To handle Vertica's [INTERVAL](#) data type, you must use JDBC's database-specific object type.

When reading an interval value from a result set, use the `ResultSet.getObject()` method to retrieve the value, and then cast it to one of the Vertica interval classes: `VerticaDayTimeInterval` (which represents all ten types of day/time intervals) or `VerticaYearMonthInterval` (which represents all three types of year/month intervals).

Note: The units interval style is not supported. Do not use the `SET INTERVALSTYLE` statement to change the interval style in your client applications.

Using Intervals in Batch Inserts

When inserting batches into tables that contain interval data, you must create instances of the `VerticaDayTimeInterval` or `VerticaYearMonthInterval` classes to hold the data you want to insert. You set values either when calling the class's constructor, or afterwards using setters. You then insert your interval values using the `PreparedStatement.setObject()` method. You can also use the `.setString()` method, passing it a string in `"DD HH:MM:SS"` or `"YY-MM"` format.

The following example demonstrates inserting data into a table containing a day/time interval and a year/month interval:

```
import java.sql.*;
import java.util.Properties;
// You need to import the Vertica JDBC classes to be able to instantiate
// the interval classes.
import com.vertica.jdbc.*;

public class IntervalDemo {
    public static void main(String[] args) {
        // If running under a Java 5 JVM, use you need to load the JDBC driver
        // using Class.forName here
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/VMart", myProp);
            // Create table for interval values
            Statement stmt = conn.createStatement();
            stmt.execute("DROP TABLE IF EXISTS interval_demo");
            stmt.executeUpdate("CREATE TABLE interval_demo("
                + "DayInt INTERVAL DAY TO SECOND, "
                + "MonthInt INTERVAL YEAR TO MONTH)");
            // Insert data into interval columns using
            // VerticaDayTimeInterval and VerticaYearMonthInterval
            // classes.
            PreparedStatement pstmt = conn.prepareStatement(
                "INSERT INTO interval_demo VALUES(?,?)");
            // Create instances of the Vertica classes that represent
            // intervals.
            VerticaDayTimeInterval dayInt = new VerticaDayTimeInterval(10, 0,
                5, 40, 0, 0, false);
```

```
VerticaYearMonthInterval monthInt = new VerticaYearMonthInterval(
    10, 6, false);
// These objects can also be manipulated using setters.
dayInt.setHour(7);
// Add the interval values to the batch
((VerticaPreparedStatement) pstmt).setObject(1, dayInt);
((VerticaPreparedStatement) pstmt).setObject(2, monthInt);
pstmt.addBatch();
// Set another row from strings.
// Set day interval in "days HH:MM:SS" format
pstmt.setString(1, "10 10:10:10");
// Set year to month value in "MM-YY" format
pstmt.setString(2, "12-09");
pstmt.addBatch();
// Execute the batch to insert the values.
try {
    pstmt.executeBatch();
} catch (SQLException e) {
    System.out.println("Error message: " + e.getMessage());
}
```

Reading Interval Values

You read an interval value from a result set using the `ResultSet.getObject()` method, and cast the object to the appropriate Vertica object class: `VerticaDayTimeInterval` for day/time intervals or `VerticaYearMonthInterval` for year/month intervals. This is easy to do if you know that the column contains an interval, and you know what type of interval it is. If your application cannot assume the structure of the data in the result set it reads in, you can test whether a column contains a database-specific object type, and if so, determine whether the object belongs to either the `VerticaDayTimeInterval` or `VerticaYearMonthInterval` classes.

```
// Retrieve the interval values inserted by previous demo.
// Query the table to get the row back as a result set.
ResultSet rs = stmt.executeQuery("SELECT * FROM interval_demo");
// If you do not know the types of data contained in the result set,
// you can read its metadata to determine the type, and use
// additional information to determine the interval type.
ResultSetMetaData md = rs.getMetaData();
while (rs.next()) {
    for (int x = 1; x <= md.getColumnCount(); x++) {
        // Get data type from metadata
        int colDataType = md.getColumnType(x);
        // You can get the type in a string:
        System.out.println("Column " + x + " is a "
            + md.getColumnTypeName(x));
        // Normally, you'd have a switch statement here to
        // handle all sorts of column types, but this example is
        // simplified to just handle database-specific types
        if (colDataType == Types.OTHER) {
            // Column contains a database-specific type. Determine
            // what type of interval it is. Assuming it is an
            // interval...
```

```
Object columnVal = rs.getObject(x);
if (columnVal instanceof VerticaDayTimeInterval) {
    // We know it is a date time interval
    VerticaDayTimeInterval interval =
        (VerticaDayTimeInterval) columnVal;
    // You can use the getters to access the interval's
    // data
    System.out.print("Column " + x + "'s value is ");
    System.out.print(interval.getDay() + " Days ");
    System.out.print(interval.getHour() + " Hours ");
    System.out.println(interval.getMinute()
        + " Minutes");
} else if (columnVal instanceof VerticaYearMonthInterval) {
    VerticaYearMonthInterval interval =
        (VerticaYearMonthInterval) columnVal;
    System.out.print("Column " + x + "'s value is ");
    System.out.print(interval.getYear() + " Years ");
    System.out.println(interval.getMonth() + " Months");
} else {
    System.out.println("Not an interval.");
}
}
}
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

The example prints the following to the console:

```
Column 1 is a INTERVAL DAY TO SECOND
Column 1's value is 10 Days 7 Hours 5 Minutes
Column 2 is a INTERVAL YEAR TO MONTH
Column 2's value is 10 Years 6 Months
Column 1 is a INTERVAL DAY TO SECOND
Column 1's value is 10 Days 10 Hours 10 Minutes
Column 2 is a INTERVAL YEAR TO MONTH
Column 2's value is 12 Years 9 Months
```

Another option is to use database metadata to find columns that contain intervals.

```
// Determine the interval data types by examining the database
// metadata.
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet dbMeta = dbmd.getColumns(null, null, "interval_demo", null);
int colcount = 0;
while (dbMeta.next()) {

    // Get the metadata type for a column.
    int javaType = dbMeta.getInt("DATA_TYPE");

    System.out.println("Column " + ++colcount + " Type name is " +
        dbMeta.getString("TYPE_NAME"));

    if(javaType == Types.OTHER) {
```

```
// The SQL_DATETIME_SUB column in the metadata tells you
// Specifically which subtype of interval you have.
// The VerticaDayTimeInterval.isDayTimeInterval()
// methods tells you if that value is a day time.
//
int intervalType = dbMeta.getInt("SQL_DATETIME_SUB");
if(VerticaDayTimeInterval.isDayTimeInterval(intervalType)) {
    // Now you know it is one of the 10 day/time interval types.
    // When you select this column you can cast to
    // VerticaDayTimeInterval.
    // You can get more specific by checking intervalType
    // against each of the 10 constants directly, but
    // they all are represented by the same object.
    System.out.println("column " + colcount + " is a " +
        "VerticaDayTimeInterval intervalType = "
        + intervalType);
} else if(VerticaYearMonthInterval.isYearMonthInterval(
    intervalType)) {
    //now you know it is one of the 3 year/month intervals,
    //and you can select the column and cast to
    // VerticaYearMonthInterval
    System.out.println("column " + colcount + " is a " +
        "VerticaDayTimeInterval intervalType = "
        + intervalType);
} else {
    System.out.println("Not an interval type.");
}
}
```

Executing Queries Through JDBC

To run a query through JDBC:

1. Connect with the Vertica database. See [Creating and Configuring a Connection](#).
2. Run the query.

The method you use to run the query depends on the type of query you want to run:

- a DDL query that does not return a result set.
- a DDL query that returns a result set.
- a DML query

Executing DDL (Data Definition Language) Queries

To run DDL queries, such as [CREATE TABLE](#) and [COPY](#), use the `Statement.execute()` method. You get an instance of this class by calling the `createStatement` method of your

connection object.

The following example creates an instance of the `Statement` class and uses it to execute a `CREATE TABLE` and a `COPY` query:

```
Statement stmt = conn.createStatement();
stmt.execute("CREATE TABLE address_book (Last_Name char(50) default ',' +
'First_Name char(50),Email char(50),Phone_Number char(50)");
stmt.execute("COPY address_book FROM 'address.dat' DELIMITER ',' NULL 'null'");
```

Executing Queries That Return Result Sets

Use the `Statement` class's `executeQuery` method to execute queries that return a result set, such as `SELECT`. To get the data from the result set, use methods such as `getInt`, `getString`, and `getDouble` to access column values depending upon the data types of columns in the result set. Use `ResultSet.next` to advance to the next row of the data set.

```
ResultSet rs = null;
rs = stmt.executeQuery("SELECT First_Name, Last_Name FROM address_book");
int x = 1;
while(rs.next()){
    System.out.println(x + ". " + rs.getString(1).trim() + " "
        + rs.getString(2).trim());
    x++;
}
```

Note: The Vertica JDBC driver does not support scrollable cursors. You can only read forwards through the result set.

Executing DML (Data Manipulation Language) Queries Using `executeUpdate`

Use the `executeUpdate` method for DML SQL queries that change data in the database, such as `INSERT`, `UPDATE` and `DELETE` which do not return a result set.

```
stmt.executeUpdate("INSERT INTO address_book " +
"VALUES ('Ben-Shachar', 'Tamar', 'tamarrow@example.com', " +
"555-380-6466'");
stmt.executeUpdate("INSERT INTO address_book (First_Name, Email) " +
"VALUES ('Pete','pete@example.com')");
```

Note: The Vertica JDBC driver's `Statement` class supports executing multiple statements in the SQL string you pass to the `execute` method. The `PreparedStatement` class does not support using multiple statements in a single execution.

Loading Data Through JDBC

You can use any of the following methods to load data via the JDBC interface:

- Executing a SQL INSERT statement to insert a single row directly.
- Batch loading data using a prepared statement.
- Bulk loading data from files or streams using [COPY](#).

When loading data into Vertica, you need to decide whether to write data to the Write Optimized Store (WOS) or the Read Optimized Store (ROS). By default, most data loading methods insert data into the WOS until it fills up, then insert any additional data directly into ROS containers (called AUTO mode). This is the best method to use when frequently loading small amounts of data (often referred to as trickle-loading). When performing less frequent large data loads (any loads over 100MB of data at once), you should change this behavior to insert data directly into the ROS.

The following sections explain in detail how you load data using JDBC.

Using a Single Row Insert

The simplest way to insert data into a table is to use the SQL [INSERT](#) statement. You can use this statement by instantiating a member of the Statement class, and use its `executeUpdate()` method to run your SQL statement.

The following code fragment demonstrates how you can create a Statement object and use it to insert data into a table named `address_book`:

```
Statement stmt = conn.createStatement();
stmt.executeUpdate("INSERT INTO address_book " +
    "VALUES ('Smith', 'John', 'jsmith@example.com', " +
    "'555-123-4567')");
```

This method has a few drawbacks: you need convert your data to string and escape any special characters in your data. A better way to insert data is to use prepared statements. See [Batch Inserts Using JDBC Prepared Statements](#).

Using LONG VARCHAR and LONG VARBINARY Data Types with JDBC

Using LONG VARCHAR and LONG VARBINARY data types in a JDBC client application is similar to using VARCHAR and VARBINARY data types. The JDBC driver transparently handles the conversion (for example, between a Java String object and a LONG VARCHAR). The following example code demonstrates inserting and retrieving a LONG VARCHAR string. It uses the JDBC Types class to determine the data type of the string returned by Vertica, although it does not actually need to know whether the database column is a LONG VARCHAR or just a VARCHAR in order to retrieve the value.

```
import java.sql.*;
import java.util.Properties;

public class LongVarcharExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.vertica.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Could not find the JDBC driver class.");
            e.printStackTrace();
            return;
        }
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",
                myProp);
            // establish connection and make a table for the data.
            Statement stmt = conn.createStatement();

            // How long we want the example string to be. This is
            // larger than can fit into a traditional VARCHAR (which is limited
            // to 65000.
            int length = 100000;

            // Create a table with a LONG VARCHAR column that can store
            // the string we want to insert.
            stmt.execute("DROP TABLE IF EXISTS longtable CASCADE");
            stmt.execute("CREATE TABLE longtable (text LONG VARCHAR(" + length
                + "))");
            // Build a long string by appending an integer to a string builder
            // until we hit the size limit. Will result in a string
            // containing 01234567890123....
            StringBuilder sb = new StringBuilder(length);
            for (int i = 0; i < length; i++)
            {
                sb.append(i % 10);
            }
        }
    }
}
```

```
String value = sb.toString();

System.out.println("String value is " + value.length() +
    " characters long.");

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO longtable (text)" +
    " VALUES(?)");
try {
    // Insert LONG VARCHAR value
    System.out.println("Inserting LONG VARCHAR value");
    pstmt.setString(1, value);
    pstmt.addBatch();
    pstmt.executeBatch();

    // Query the table we created to get the value back.
    ResultSet rs = null;
    rs = stmt.executeQuery("SELECT * FROM longtable");

    // Get metadata about the result set.
    ResultSetMetaData rsmd = rs.getMetaData();
    // Print the type of the first column. Should be
    // LONG VARCHAR. Also check it against the Types class, to
    // recognize it programmatically.
    System.out.println("Column #1 data type is: " +
        rsmd.getColumnTypeName(1));
    if (rsmd.getColumnType(1) == Types.LONGVARCHAR) {
        System.out.println("It is a LONG VARCHAR");
    } else {
        System.out.println("It is NOT a LONG VARCHAR");
    }

    // Print out the string length of the returned value.
    while (rs.next()) {
        // Use the same getString method to get the value that you
        // use to get the value of a VARCHAR.
        System.out.println("Returned string length: " +
            rs.getString(1).length());
    }
} catch (SQLException e) {
    System.out.println("Error message: " + e.getMessage());
    return; // Exit if there was an error
}
// Cleanup
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the

octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

Batch Inserts Using JDBC Prepared Statements

You can load batches of data into Vertica using prepared **INSERT** statements—server-side statements that you set up once, and then call repeatedly. You instantiate a member of the `PreparedStatement` class with a SQL statement that contains question mark placeholders for data. For example:

```
PreparedStatement pstmt = conn.prepareStatement(  
    "INSERT INTO customers(last, first, id) VALUES(?,?,?)");
```

You then set the parameters using data-type-specific methods on the `PreparedStatement` object, such as `setString()` and `setInt()`. Once your parameters are set, call the `addBatch()` method to add the row to the batch. When you have a complete batch of data ready, call the `executeBatch()` method to execute the insert batch.

Behind the scenes, the batch insert is converted into a **COPY** statement. When the connection's `AutoCommit` parameter is disabled, Vertica keeps the **COPY** statement open and uses it to load subsequent batches until the transaction is committed, the cursor is closed, or your application executes anything else (or executes any statement using another `Statement` or `PreparedStatement` object). Using a single **COPY** statement for multiple batch inserts makes loading data more efficient. If you are loading multiple batches, you should disable the `AutoCommit` property of the database to take advantage of this increased efficiency.

When performing batch inserts, experiment with various batch and row sizes to determine the settings that provide the best performance.

The following example demonstrates using a prepared statement to batch insert data.

```
import java.sql.*;  
import java.util.Properties;  
  
public class BatchInsertExample {  
    public static void main(String[] args) {  
        Properties myProp = new Properties();  
        myProp.put("user", "ExampleUser");  
        myProp.put("password", "password123");  
  
        //Set streamingBatchInsert to True to enable streaming mode for batch inserts.  
        //myProp.put("streamingBatchInsert", "True");  
  
        Connection conn;  
        try {  
            conn = DriverManager.getConnection(  
                "jdbc:vertica://VerticaHost:5433/ExampleDB",  
                myProp);  
            // establish connection and make a table for the data.  
            Statement stmt = conn.createStatement();
```

```
// Set AutoCommit to false to allow Vertica to reuse the same
// COPY statement
conn.setAutoCommit(false);

// Drop table and recreate.
stmt.execute("DROP TABLE IF EXISTS customers CASCADE");
stmt.execute("CREATE TABLE customers (CustID int, Last_Name"
    + " char(50), First_Name char(50),Email char(50), "
    + "Phone_Number char(12))");
// Some dummy data to insert.
String[] firstNames = new String[] { "Anna", "Bill", "Cindy",
    "Don", "Eric" };
String[] lastNames = new String[] { "Allen", "Brown", "Chu",
    "Dodd", "Estavez" };
String[] emails = new String[] { "aang@example.com",
    "b.brown@example.com", "cindy@example.com",
    "d.d@example.com", "e.estavez@example.com" };
String[] phoneNumbers = new String[] { "123-456-7890",
    "555-444-3333", "555-867-5309",
    "555-555-1212", "781-555-0000" };
// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO customers (CustID, Last_Name, " +
    "First_Name, Email, Phone_Number)" +
    " VALUES(?,?,?,?,?)");
// Add rows to a batch in a loop. Each iteration adds a
// new row.
for (int i = 0; i < firstNames.length; i++) {
    // Add each parameter to the row.
    pstmt.setInt(1, i + 1);
    pstmt.setString(2, lastNames[i]);
    pstmt.setString(3, firstNames[i]);
    pstmt.setString(4, emails[i]);
    pstmt.setString(5, phoneNumbers[i]);
    // Add row to the batch.
    pstmt.addBatch();
}

try {
    // Batch is ready, execute it to insert the data
    pstmt.executeBatch();
} catch (SQLException e) {
    System.out.println("Error message: " + e.getMessage());
    return; // Exit if there was an error
}

// Commit the transaction to close the COPY command
conn.commit();

// Print the resulting table.
ResultSet rs = null;
rs = stmt.executeQuery("SELECT CustID, First_Name, "
    + "Last_Name FROM customers ORDER BY CustID");
while (rs.next()) {
    System.out.println(rs.getInt(1) + " - "
        + rs.getString(2).trim() + " "
        + rs.getString(3).trim());
}
}
```

```
// Cleanup
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

The result of running the example code is:

```
1 - Anna Allen
2 - Bill Brown
3 - Cindy Chu
4 - Don Dodd
5 - Eric Estavez
```

Streaming Batch Inserts

By default, Vertica performs batch inserts by caching each row and inserting the cache when the user calls the `executeBatch()` method. Vertica also supports streaming batch inserts. A streaming batch insert adds a row to the database each time the user calls `addBatch()`. Streaming batch inserts improve database performance by allowing parallel processing and reducing memory demands.

Note: Once you begin a streaming batch insert, you cannot make other JDBC calls that require client-server communication until you have executed the batch or closed or rolled back the connection.

To enable streaming batch inserts, set the `streamingBatchInsert` property to `True`. The preceding code sample includes a line enabling `streamingBatchInsert` mode. Remove the `//` comment marks to enable this line and activate streaming batch inserts.

The following table explains the various batch insert methods and how their behavior differs between default batch insert mode and streaming batch insert mode.

Method	Default Batch Insert Behavior	Streaming Batch Insert Behavior
<code>addBatch()</code>	Adds a row to the row cache.	Inserts a row into the database.
<code>executeBatch()</code>	Adds the contents of the row cache to the database in a single action.	Sends an end-of-batch message to the server and returns an array of integers indicating the success or failure of each <code>addBatch()</code> attempt.

<code>clearBatch()</code>	Clears the row cache without inserting any rows.	Not supported. Triggers an exception if used when streaming batch inserts are enabled.
---------------------------	--	--

Notes

- Using the `PreparedStatement.setFloat()` method can cause rounding errors. If precision is important, use the `.setDouble()` method instead.
- The `PreparedStatement` object caches the connection's `AutoCommit` property when the statement is prepared. Later changes to the `AutoCommit` property have no effect on the prepared statement.

Loading Batches Directly into ROS

When loading large batches of data (more than 100MB or so), you should load the data directly into ROS containers. Inserting directly into ROS is more efficient for large loads than AUTO mode, since it avoids overflowing the WOS and spilling the remainder of the batch to ROS. Otherwise, the Tuple Mover has to perform a moveout on the data in the WOS, while subsequent data is directly written into ROS containers causing your data to be segmented across storage containers.

When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see [Choosing a Load Method](#).

To directly load batches into ROS, set the `directBatchInsert` connection property to true. See [Setting and Getting Connection Property Values](#) for an explanation of how to set connection properties. When this property is set to true, all batch inserts bypass the WOS and load directly into a ROS container.

If all of batches being inserted using a connection should be inserted into the ROS, you should set the `DirectBatchInsert` connection property to true in the `Properties` object you use to create the connection:

```
Properties myProp = new Properties();
myProp.put("user", "ExampleUser");
myProp.put("password", "password123");
// Enable directBatchInsert for this connection
myProp.put("DirectBatchInsert", "true");
Connection conn;
try {
    conn = DriverManager.getConnection(
        "jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
    ...
}
```

If you will be using the connection for inserting both large and small batches (or you do not know the size batches you will be inserting when you create the `Connection` object), you can set the `DirectBatchInsert` property after the connection has been established using the `VerticaConnection.setProperty` method:

```
((VerticaConnection)conn).setProperty("DirectBatchInsert", true);
```

See [Setting and Getting Connection Property Values](#) for a full example of setting `DirectBatchInsert`.

Error Handling During Batch Loads

When loading individual batches, you can find how many rows were accepted and what rows were rejected (see [Identifying Accepted and Rejected Rows](#) for details). If you have disabled the `AutoCommit` connection setting, other errors (such as disk space errors, for example) do not occur while inserting individual batches. This behavior is caused by having a single SQL `COPY` statement perform the loading of multiple consecutive batches (which makes the load process more efficient). It is only when the `COPY` statement closes that the batched data is committed and Vertica reports other types of errors.

Therefore, your bulk loading application should be prepared to check for errors when the `COPY` statement closes. You can trigger the `COPY` statement to close by:

- ending the batch load transaction by calling `Connection.commit()`
- closing the statement using `Statement.close()`
- setting the connection's `AutoCommit` property to `true` before inserting the last batch in the load

Note: The `COPY` statement also closes if you execute any non-insert statement or execute any statement using a different `Statement` or `PreparedStatement` object. Ending the `COPY` statement using either of these methods can lead to confusion and a harder-to-maintain application, since you would need to handle batch load errors in a non-batch load statement. You should explicitly end the `COPY` statement at the end of your batch load and handle any errors at that time.

Identifying Accepted and Rejected Rows (JDBC)

The return value of `PreparedStatement.executeBatch` is an integer array containing the success or failure status of inserting each row. A value `1` means the row was accepted and a value of `-3` means that the row was rejected. In the case where an exception occurred during

the batch execution, you can also get the array using `BatchUpdateException.getUpdateCounts()`.

The following example extends the example shown in [Batch Inserts Using JDBC Prepared Statements](#) to retrieve this array and display the results the batch load.

```
import java.sql.*;
import java.util.Arrays;
import java.util.Properties;

public class BatchInsertErrorHandlingExample {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;

        // establish connection and make a table for the data.
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",
                myProp);

            // Disable auto commit
            conn.setAutoCommit(false);

            // Create a statement
            Statement stmt = conn.createStatement();
            // Drop table and recreate.
            stmt.execute("DROP TABLE IF EXISTS customers CASCADE");
            stmt.execute("CREATE TABLE customers (CustID int, Last_Name"
                + " char(50), First_Name char(50), Email char(50), "
                + "Phone_Number char(12))");

            // Some dummy data to insert. The one row won't insert because
            // the phone number is too long for the phone column.
            String[] firstNames = new String[] { "Anna", "Bill", "Cindy",
                "Don", "Eric" };
            String[] lastNames = new String[] { "Allen", "Brown", "Chu",
                "Dodd", "Estavez" };
            String[] emails = new String[] { "aang@example.com",
                "b.brown@example.com", "cindy@example.com",
                "d.d@example.com", "e.estavez@example.com" };
            String[] phoneNumbers = new String[] { "123-456-789",
                "555-444-3333", "555-867-53093453453",
                "555-555-1212", "781-555-0000" };

            // Create the prepared statement
            PreparedStatement pstmt = conn.prepareStatement(
                "INSERT INTO customers (CustID, Last_Name, " +
                "First_Name, Email, Phone_Number) +
                " VALUES(?, ?, ?, ?, ?)");

            // Add rows to a batch in a loop. Each iteration adds a
            // new row.
            for (int i = 0; i < firstNames.length; i++) {
                // Add each parameter to the row.
                pstmt.setInt(1, i + 1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
pstmt.setString(2, lastNames[i]);
pstmt.setString(3, firstNames[i]);
pstmt.setString(4, emails[i]);
pstmt.setString(5, phoneNumbers[i]);
// Add row to the batch.
pstmt.addBatch();
}

// Integer array to hold the results of inserting
// the batch. Will contain an entry for each row,
// indicating success or failure.
int[] batchResults = null;

try {
    // Batch is ready, execute it to insert the data
    batchResults = pstmt.executeBatch();
} catch (BatchUpdateException e) {
    // We expect an exception here, since one of the
    // inserted phone numbers is too wide for its column. All of the
    // rest of the rows will be inserted.
    System.out.println("Error message: " + e.getMessage());

    // Batch results isn't set due to exception, but you
    // can get it from the exception object.
    //
    // In your own code, you shouldn't assume the a batch
    // exception occurred, since exceptions can be thrown
    // by the server for a variety of reasons.
    batchResults = e.getUpdateCounts();
}
// You should also be prepared to catch SQLExceptions in your own
// application code, to handle dropped connections and other general
// problems.

// Commit the transaction
conn.commit();

// Print the array holding the results of the batch insertions.
System.out.println("Return value from inserting batch: "
    + Arrays.toString(batchResults));
// Print the resulting table.
ResultSet rs = null;
rs = stmt.executeQuery("SELECT CustID, First_Name, "
    + "Last_Name FROM customers ORDER BY CustID");
while (rs.next()) {
    System.out.println(rs.getInt(1) + " - "
        + rs.getString(2).trim() + " "
        + rs.getString(3).trim());
}

// Cleanup
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

Running the above example produces the following output on the console:

```
Error message: [Vertica][VJDBC](100172) One or more rows were rejected by the server.Return value
from inserting batch: [1, 1, -3, 1, 1]
1 - Anna Allen
2 - Bill Brown
4 - Don Dodd
5 - Eric Estavez
```

Notice that the third row failed to insert because its phone number is too long for the Phone_Number column. All of the rest of the rows in the batch (including those after the error) were correctly inserted.

Note: It is more efficient for you to ensure that the data you are inserting is the correct data type and width for the table column you are inserting it into than to handle exceptions after the fact.

Rolling Back Batch Loads on the Server

Batch loads always insert all of their data, even if one or more rows is rejected. Only the rows that caused errors in a batch are not loaded. When the database connection's `AutoCommit` property is true, batches automatically commit their transactions when they complete, so once the batch finishes loading, the data is committed.

In some cases, you may want all of the data in a batch to be successfully inserted—none of the data should be committed if an error occurs. The best way to accomplish this is to turn off the database connection's `AutoCommit` property to prevent batches from automatically committing themselves. Then, if a batch encounters an error, you can roll back the transaction after catching the `BatchUpdateException` caused by the insertion error.

The following example demonstrates performing a rollback if any error occurs when loading a batch.

```
import java.sql.*;
import java.util.Arrays;
import java.util.Properties;

public class RollbackBatchOnError {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",
                myProp);
            // Disable auto-commit. This will allow you to roll back a
            // a batch load if there is an error.
            conn.setAutoCommit(false);
            // establish connection and make a table for the data.
```

```
Statement stmt = conn.createStatement();
// Drop table and recreate.
stmt.execute("DROP TABLE IF EXISTS customers CASCADE");
stmt.execute("CREATE TABLE customers (CustID int, Last_Name"
    + " char(50), First_Name char(50),Email char(50), "
    + "Phone_Number char(12))");

// Some dummy data to insert. The one row won't insert because
// the phone number is too long for the phone column.
String[] firstNames = new String[] { "Anna", "Bill", "Cindy",
    "Don", "Eric" };
String[] lastNames = new String[] { "Allen", "Brown", "Chu",
    "Doddd", "Estavez" };
String[] emails = new String[] { "aang@example.com",
    "b.brown@example.com", "cindy@example.com",
    "d.d@example.com", "e.estavez@example.com" };
String[] phoneNumbers = new String[] { "123-456-789",
    "555-444-3333", "555-867-53094535", "555-555-1212",
    "781-555-0000" };

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO customers (CustID, Last_Name, " +
    "First_Name, Email, Phone_Number) "+
    "VALUES(?,?,?,?,?)");

// Add rows to a batch in a loop. Each iteration adds a
// new row.
for (int i = 0; i < firstNames.length; i++) {
    // Add each parameter to the row.
    pstmt.setInt(1, i + 1);
    pstmt.setString(2, lastNames[i]);
    pstmt.setString(3, firstNames[i]);
    pstmt.setString(4, emails[i]);
    pstmt.setString(5, phoneNumbers[i]);
    // Add row to the batch.
    pstmt.addBatch();
}

// Integer array to hold the results of inserting
// the batch. Will contain an entry for each row,
// indicating success or failure.
int[] batchResults = null;
try {
    // Batch is ready, execute it to insert the data
    batchResults = pstmt.executeBatch();
    // If we reach here, we inserted the batch without errors.
    // Commit it.
    System.out.println("Batch insert successful. Committing.");
    conn.commit();
} catch (BatchUpdateException e) {
    System.out.println("Error message: " + e.getMessage());
    // Batch results isn't set due to exception, but you
    // can get it from the exception object.
    batchResults = e.getUpdateCounts();
    // Roll back the batch transaction.
    System.out.println("Rolling back batch insertion");
    conn.rollback();
}
catch (SQLException e) {
    // General SQL errors, such as connection issues, throw
    // SQLExceptions. Your application should do something more
    // than just print a stack trace,
```

```
        e.printStackTrace();
    }
    System.out.println("Return value from inserting batch: "
        + Arrays.toString(batchResults));
    System.out.println("Customers table contains:");

    // Print the resulting table.
    ResultSet rs = null;
    rs = stmt.executeQuery("SELECT CustID, First_Name, "
        + "Last_Name FROM customers ORDER BY CustID");
    while (rs.next()) {
        System.out.println(rs.getInt(1) + " - "
            + rs.getString(2).trim() + " "
            + rs.getString(3).trim());
    }

    // Cleanup
    conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

Running the above example prints the following on the system console:

```
Error message: [Vertica][VJDBC](100172) One or more rows were rejected by the server.Rolling back
batch insertion
Return value from inserting batch: [1, 1, -3, 1, 1]
Customers table contains:
```

The return values indicate whether each row was successfully inserted. The value 1 means the row inserted without any issues, and a -3 indicates the row failed to insert.

The customers table is empty since the batch insert was rolled back due to the error caused by the third column.

Bulk Loading Using the COPY Statement

One of the fastest ways to load large amounts of data into Vertica at once (bulk loading) is to use the [COPY statement](#). This statement loads data from a file stored on a Vertica host (or in a data stream) into a table in the database. You can pass the COPY statement parameters that define the format of the data in the file, how the data is to be transformed as it is loaded, how to handle errors, and how the data should be loaded. See the [COPY](#) documentation in the SQL Reference Manual for details.

One parameter that is particularly important is the `DIRECT` option, which tells COPY to load the data directly into ROS rather than going through the WOS. You should use this option when you are loading large files (over 100MB) into the database. Without this option, your load may

fill the WOS and overflow into ROS, requiring the [Tuple Mover](#) to perform a Moveout on the data in the WOS. It is more efficient to directly load into ROS and avoid forcing a moveout.

Only a superuser can use the COPY statement to copy a file stored on a host, so you must connect to the database using a superuser account. If you want to have a non-superuser user bulk-load data, you can use COPY to load from a stream on the host (such as STDIN) rather than a file or stream data from the client (see [Streaming Data Via JDBC](#)). You can also perform a standard [batch insert using a prepared statement](#), which uses the COPY statement in the background to load the data.

Note: When using this [COPY parameter](#) on any node, confirm that the source file is identical on all nodes. Using different files can produce inconsistent results.

The following example demonstrates using the COPY statement through the JDBC to load a file name `customers.txt` into a new database table. This file must be stored on the database host to which your application connects (in this example, a host named `VerticaHost`). Since the `customers.txt` file used in the example is very large, this example uses the `DIRECT` option to bypass WOS and load directly into ROS.

```
import java.sql.*;
import java.util.Properties;
import com.vertica.jdbc.*;

public class COPYFromFile {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleAdmin"); // Must be superuser
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",myProp);
            // Disable AutoCommit
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            // Create a table to hold data.
            stmt.execute("DROP TABLE IF EXISTS customers;");
            stmt.execute("CREATE TABLE IF NOT EXISTS customers (Last_Name char(50) "
                + "NOT NULL, First_Name char(50),Email char(50), "
                + "Phone_Number char(15))");

            // Use the COPY command to load data. Load directly into ROS, since
            // this load could be over 100MB. Use ENFORCELENGTH to reject
            // strings too wide for their columns.
            boolean result = stmt.execute("COPY customers FROM "
                + "'/data/customers.txt' DIRECT ENFORCELENGTH");

            // Determine if execution returned a count value, or a full result
            // set.
            if (result) {
                System.out.println("Got result set");
            } else {
                // Count will usually return the count of rows inserted.
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
System.out.println("Got count");
int rowCount = stmt.getUpdateCount();
System.out.println("Number of accepted rows = " + rowCount);
}

// Commit the data load
conn.commit();
} catch (SQLException e) {
System.out.print("Error: ");
System.out.println(e.toString());
}
}
}
```

The example prints the following out to the system console when run (assuming that the `customers.txt` file contained two million valid rows):

```
Number of accepted rows = 2000000
```

Streaming Data Via JDBC

There are two options to stream data from a file on the client to your Vertica database:

- Use the `VerticaCopyStream` class to stream data in an object-oriented manner - details on the class are available in the [JDBC Documentation](#)
- Execute a [COPY LOCAL](#) SQL statement to stream the data

The topics in this section explain how to use these options.

Using VerticaCopyStream

The `VerticaCopyStream` class (details on the class are available in the [JDBC Documentation](#)) lets you stream data from the client system to a Vertica database. It lets you use the SQL [COPY statement](#) directly without having to copy the data to a host in the database cluster first. Using the `COPY` command to load data from the host requires superuser privileges to be able to access the host's filesystem. The `COPY` statement used to load data from a stream does not require superuser privileges so your client can connect using any user account that has `INSERT` privileges on the table that will receive the data.

To copy streams into the database:

1. Disable the database connections AutoCommit connection parameter.
2. Instantiate a `VerticaCopyStreamObject`, passing it at least the database connection objects and a string containing a COPY statement to load the data. This statement **must** copy data from the STDIN into your table. You can use whatever parameters are appropriate for your data load.

Note: The `VerticaCopyStreamObject` constructor optionally takes a single `InputStream` object, or a `List` of `InputStream` objects. This option lets you pre-populate the list of streams to be copied into the database.

3. Call `VerticaCopyStreamObject.start()` to start the COPY statement and begin streaming the data in any streams you have already added to the `VerticaCopyStreamObject`.
4. Call `VerticaCopyStreamObject.addStream()` to add additional streams to the list of streams to send to the database. You can then call `VerticaCopyStreamObject.execute()` to stream them to the server.
5. Optionally, call `VerticaCopyStreamObject.getRejects()` to get a list of rejected rows from the last `.execute()` call. The list of rejects is reset by each call to `.execute()` or `.finish()`.

Note: If you used either the REJECTED DATA or EXCEPTIONS options in the COPY statement you passed to `VerticaCopyStreamObject` the object in step 2, `.getRejects()` returns an empty list. You can only use one method of tracking the rejected rows at a time.

6. When you are finished adding streams, call `VerticaCopyStreamObject.finish()` to send any remaining streams to the database and close the COPY statement.
7. Call `Connection.commit()` to commit the loaded data.

Getting Rejected Rows

The `VerticaCopyStreamObject.getRejects()` method returns a `List` containing the row numbers of rows that were rejected after the previous `.execute()` method call. Each call to `.execute()` clears the list of rejected rows, so you need to call `.getRejects()` after each call to `.execute()`. Since `.start()` and `.finish()` also call `.execute()` to send any pending streams to the server, you should also call `.getRejects()` after these methods as well.

The following example demonstrates loading the content of five text files stored on the client system into a table.

```
import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;
import com.vertica.jdbc.VerticaConnection;
import com.vertica.jdbc.VerticaCopyStream;

public class CopyMultipleStreamsExample {
    public static void main(String[] args) {
        // Note: If running on Java 5, you need to call Class.forName
        // to manually load the JDBC driver.
        // Set up the properties of the connection
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser"); // Must be superuser
        myProp.put("password", "password123");
        // When performing bulk loads, you should always disable the
        // connection's AutoCommit property to ensure the loads happen as
        // efficiently as possible by reusing the same COPY command and
        // transaction.
        myProp.put("AutoCommit", "false");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            Statement stmt = conn.createStatement();

            // Create a table to receive the data
            stmt.execute("DROP TABLE IF EXISTS customers");
            stmt.execute("CREATE TABLE customers (Last_Name char(50), "
                + "First_Name char(50),Email char(50), "
                + "Phone_Number char(15))");

            // Prepare the query to insert from a stream. This query must use
            // the COPY statement to load data from STDIN. Unlike copying from
            // a file on the host, you do not need superuser privileges to
            // copy a stream. All your user account needs is INSERT privileges
            // on the target table.
            String copyQuery = "COPY customers FROM STDIN "
```

```
+ "DELIMITER '|' DIRECT ENFORCELENGTH";

// Create an instance of the stream class. Pass in the
// connection and the query string.
VerticaCopyStream stream = new VerticaCopyStream(
    (VerticaConnection) conn, copyQuery);

// Keep running count of the number of rejects
int totalRejects = 0;

// start() starts the stream process, and opens the COPY command.
stream.start();

// If you added streams to VerticaCopyStream before calling start(),
// You should check for rejects here (see below). The start() method
// calls execute() to send any pre-queued streams to the server
// once the COPY statement has been created.

// Simple for loop to load 5 text files named customers-1.txt to
// customers-5.txt
for (int loadNum = 1; loadNum <= 5; loadNum++) {
    // Prepare the input file stream. Read from a local file.
    String filename = "C:\\Data\\customers-" + loadNum + ".txt";
    System.out.println("\n\nLoading file: " + filename);
    File inputFile = new File(filename);
    FileInputStream inputStream = new FileInputStream(inputFile);

    // Add stream to the VerticaCopyStream
    stream.addStream(inputStream);

    // call execute() to load the newly added stream. You could
    // add many streams and call execute once to load them all.
    // Which method you choose depends mainly on whether you want
    // the ability to check the number of rejections as the load
    // progresses so you can stop if the number of rejects gets too
    // high. Also, high numbers of InputStreams could create a
    // resource issue on your client system.
    stream.execute();

    // Show any rejects from this execution of the stream load
    // getRejects() returns a List containing the
    // row numbers of rejected rows.
    List<Long> rejects = stream.getRejects();

    // The size of the list gives you the number of rejected rows.
    int numRejects = rejects.size();
    totalRejects += numRejects;
    System.out.println("Number of rows rejected in load #"
        + loadNum + ": " + numRejects);

    // List all of the rows that were rejected.
    Iterator<Long> rejIt = rejects.iterator();
    long linecount = 0;
    while (rejIt.hasNext()) {
        System.out.print("Rejected row #" + ++linecount);
        System.out.println(" is row " + rejIt.next());
    }
}
// Finish closes the COPY command. It returns the number of
// rows inserted.
```

```
long results = stream.finish();
System.out.println("Finish returned " + results);

// If you added any streams that hadn't been executed(),
// you should also check for rejects here, since finish()
// calls execute() to

// You can also get the number of rows inserted using
// getRowCount().
System.out.println("Number of rows accepted: "
    + stream.getRowCount());
System.out.println("Total number of rows rejected: " + totalRejects);

// Commit the loaded data
conn.commit();

} catch (Exception e) {
    e.printStackTrace();
}
}
```

Running the above example on some sample data results in the following output:

```
Loading file: C:\Data\customers-1.txtNumber of rows rejected in load #1: 3
Rejected row #1 is row 3
Rejected row #2 is row 7
Rejected row #3 is row 51
Loading file: C:\Data\customers-2.txt
Number of rows rejected in load #2: 5Rejected row #1 is row 4143
Rejected row #2 is row 6132
Rejected row #3 is row 9998
Rejected row #4 is row 10000
Rejected row #5 is row 10050
Loading file: C:\Data\customers-3.txt
Number of rows rejected in load #3: 9
Rejected row #1 is row 14142
Rejected row #2 is row 16131
Rejected row #3 is row 19999
Rejected row #4 is row 20001
Rejected row #5 is row 20005
Rejected row #6 is row 20049
Rejected row #7 is row 20056
Rejected row #8 is row 20144
Rejected row #9 is row 20236
Loading file: C:\Data\customers-4.txt
Number of rows rejected in load #4: 8
Rejected row #1 is row 23774
Rejected row #2 is row 24141
Rejected row #3 is row 25906
Rejected row #4 is row 26130
Rejected row #5 is row 27317
Rejected row #6 is row 28121
Rejected row #7 is row 29321
Rejected row #8 is row 29998
Loading file: C:\Data\customers-5.txt
Number of rows rejected in load #5: 1
Rejected row #1 is row 39997
```

```
Finish returned 39995  
Number of rows accepted: 39995  
Total number of rows rejected: 26
```

Note: The above example shows a simple load process that targets one node in the Vertica cluster. It is more efficient to simultaneously load multiple streams to multiple database nodes. Doing so greatly improves performance because it spreads the processing for the load across the cluster.

Using COPY LOCAL with JDBC

To use COPY LOCAL with JDBC, just execute a [COPY LOCAL](#) statement with the path to the source file on the client system. This method is simpler than using the `VerticaCopyStream` class (details on the class are available in the [JDBC Documentation](#)). However, you may prefer using `VerticaCopyStream` if you have many files to copy to the database or if your data comes from a source other than a file (streamed over a network connection, for example).

The following example code demonstrates using COPY LOCAL to copy a file from the client to the database. It is the same as the code shown in [Bulk Loading Using the COPY Statement](#), except for the use of the LOCAL option in the COPY statement, and the path to the data file is on the client system, rather than on the server.

Note: The exceptions/rejections files are created on the client machine when the exceptions and rejected data modifiers are specified on the copy local command. Specify a local path and filename for these modifiers when executing a COPY LOCAL query from the driver.

```
import java.sql.*;  
import java.util.Properties;  
  
public class COPYLocal {  
    public static void main(String[] args) {  
        // Note: If using Java 5, you must call Class.forName to load the  
        // JDBC driver.  
        Properties myProp = new Properties();  
        myProp.put("user", "ExampleUser"); // Do not need to superuser  
        myProp.put("password", "password123");  
        Connection conn;  
        try {  
            conn = DriverManager.getConnection(  
                "jdbc:vertica://VerticaHost:5433/ExampleDB",myProp);  
            // Disable AutoCommit  
            conn.setAutoCommit(false);  
            Statement stmt = conn.createStatement();  
            // Create a table to hold data.  
            stmt.execute("DROP TABLE IF EXISTS customers;");  
            stmt.execute("CREATE TABLE IF NOT EXISTS customers (Last_Name char(50) "
```

```
+ "NOT NULL, First_Name char(50),Email char(50), "  
+ "Phone_Number char(15))");  
  
// Use the COPY command to load data. Load directly into ROS, since  
// this load could be over 100MB. Use ENFORCELENGTH to reject  
// strings too wide for their columns.  
boolean result = stmt.execute("COPY customers FROM LOCAL "  
    + "'C:\\Data\\customers.txt' DIRECT ENFORCELENGTH");  
  
// Determine if execution returned a count value, or a full result  
// set.  
if (result) {  
    System.out.println("Got result set");  
} else {  
    // Count will usually return the count of rows inserted.  
    System.out.println("Got count");  
    int rowCount = stmt.getUpdateCount();  
    System.out.println("Number of accepted rows = " + rowCount);  
}  
  
conn.close();  
} catch (SQLException e) {  
    System.out.print("Error: ");  
    System.out.println(e.toString());  
}  
}  
}
```

The result of running this code appears below. In this case, the customers.txt file contains 10000 rows, seven of which get rejected because they contain data too wide to fit into their database columns.

```
Got countNumber of accepted rows = 9993
```

Handling Errors

When the Vertica JDBC driver encounters an error, it throws a `SQLException` or one of its subclasses. The specific subclass it throws depends on the type of error that has occurred. Most of the JDBC method calls can result in several different types of errors, in response to which the JDBC driver throws a specific `SQLException` subclass. Your client application can choose how to react to the error based on the specific exception that the JDBC driver threw.

Note: The specific `SQLException` subclasses were introduced in the JDBC 4.0 standard. If your client application runs in a Java 5 JVM, it will use the older JDBC 3.0-compliant driver which lacks these subclasses. In that case, all errors throw a `SQLException`.

The hierarchy of `SQLException` subclasses is arranged to help your client application determine what actions it can take in response to an error condition. For example:

- The JDBC driver throws `SQLTransientException` subclasses when the cause of the error may be a temporary condition, such as a timeout error (`SQLTimeoutException`) or a connection issue (`SQLTransientConnectionIssue`). Your client application can choose to retry the operation without making any sort of attempt to remedy the error, since it may not reoccur.
- The JDBC driver throws `SQLNonTransientException` subclasses when the client needs to take some action before it could retry the operation. For example, executing a statement with a SQL syntax error results in the JDBC driver throwing the a `SQLSyntaxErrorException` (a subclass of `SQLNonTransientException`). Often, your client application just has to report these errors back to the user and have him or her resolve them. For example, if the user supplied your application with a SQL statement that triggered a `SQLSyntaxErrorException`, it could prompt the user to fix the SQL error.

See [Vertica Analytics Platform SQLState Mapping to Java Exception Classes](#) for a list Java exceptions thrown by the JDBC driver.

Vertica Analytics Platform SQLState Mapping to Java Exception Classes

SQLSTATE Class or Value	Description	Java Exception Class
Class 00	Successful Completion	SQLException
Class 01	Warning	SQLWarning
Class 02	No Data	SQLException
Class 03	SQL Statement Not Yet Complete	SQLException
Class 08	Client Connection Exception	SQLNonTransientConnectionException
Class 09	Triggered Action Exception	SQLException
Class 0A	Feature Not Supported	SQLFeatureNotSupportedException
Class 0B	Invalid Transaction Initiation	SQLException
Class 0F	Locator Exception	SQLException
Class 0L	Invalid Grantor	SQLException
Class 0P	Invalid Role Specification	SQLException
Class 21	Cardinality Violation	SQLException
Class 22	Data Exception	SQLDataException
22V21	ERRCODE_INVALID_EPOCH	SQLNonTransientException

SQLSTATE Class or Value	Description	Java Exception Class
Class 23	Integrity Constraint Violation	SQLIntegrityConstraintViolationException
Class 24	Invalid Cursor State	SQLException
Class 25	Invalid Transaction State	SQLTransactionRollbackException
Class 26	Invalid SQL Statement Name	SQLException
Class 27	Triggered Data Change Violation	SQLException
Class 28	Invalid Authorization Specification	SQLInvalidAuthorizationException
Class 2B	Dependent Privilege Descriptors Still Exist	SQLDataException
Class 2D	Invalid Transaction Termination	SQLException
Class 2F	SQL Routine Exception	SQLException
Class 34	Invalid Cursor Name	SQLException
Class 38	External Routine Exception	SQLException
Class 39	External Routine Invocation Exception	SQLException

SQLSTATE Class or Value	Description	Java Exception Class
Class 3B	Savepoint Exception	SQLException
Class 3D	Invalid Catalog Name	SQLException
Class 3F	Invalid Schema Name	SQLException
Class 40	Transaction Rollback	SQLTransactionRollbackException
Class 42	Syntax Error or Access Rule Violation	SQLSyntaxErrorException
Class 44	WITH CHECK OPTION Violation	SQLException
Class 53	Insufficient Resources	SQLTransientException
53300	ERRCODE_TOO_ MANY_ CONNECTIONS	SQLNonTransientConnectionException
Class 54	Program Limit Exceeded	SQLNonTransientException
Class 55	Object Not In Prerequisite State	SQLNonTransientException
55V03	ERRCODE_LOCK_ NOT_AVAILABLE	SQLTransactionRollbackException
Class 57	Operator Intervention	SQLTransientException
57V01	ERRCODE_ ADMIN_ SHUTDOWN	SQLNonTransientConnectionException

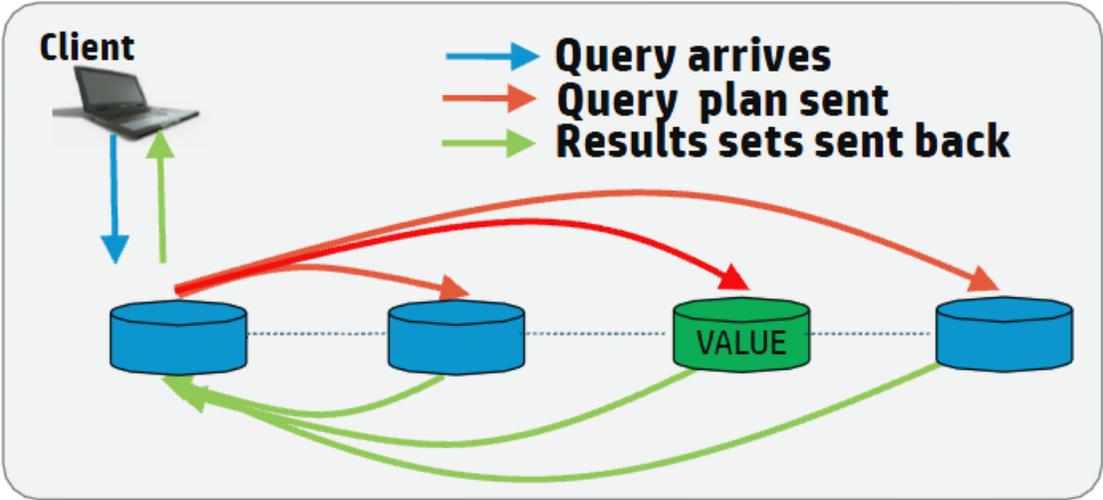
SQLSTATE Class or Value	Description	Java Exception Class
57V02	ERRCODE_CRASH_SHUTDOWN	SQLNonTransientConnectionException
57V03	ERRCODE_CANNOT_CONNECT_NOW	SQLNonTransientConnectionException
Class 58	System Error	SQLException
Class V1	Vertica-specific multi-node errors class	SQLException
Class V2	Vertica-specific miscellaneous errors class	SQLException
V2000	ERRCODE_AUTH_FAILED	SQLInvalidAuthorizationException
Class VC	Configuration File Error	SQLNonTransientException
Class VD	DB Designer errors	SQLNonTransientException
Class VP	User procedure errors	SQLNonTransientException
Class VX	Internal Error	SQLException

Routing JDBC Queries Directly to a Single Node

The JDBC driver has the ability to route queries directly to a single node using a special connection called a Routable Connection. This feature is ideal for high-volume "short" requests that return a small number of results that all exist on a single node. The common scenario for using this feature is to do high-volume lookups on data that is identified with a unique key. Routable queries typically provide lower latency and use less system resources than distributed queries. However, the data being queried must be segmented in such a way that the JDBC client can determine on which node the data resides.

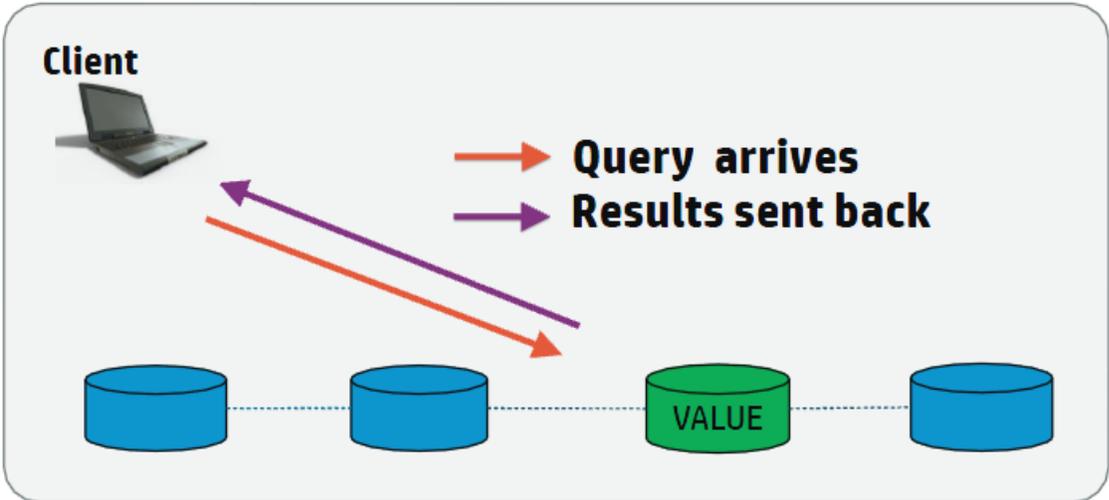
Vertica Typical Analytic Query

Typical analytic queries require dense computation on data across all nodes in the cluster and benefit from having all nodes involved in the planning and execution of the queries.



Vertica Routable Query API Query

For high-volume queries that return a single or a few rows of data, it is more efficient to execute the query on the single node that contains the data.



To effectively route a request to a single node, the client must determine the specific node on which the data resides. For the client to be able to determine the correct node, the table must be [segmented](#) by one or more columns. For example, if you segment a table on a Primary Key (PK) column, then the client can determine on which node the data resides based on the Primary Key and directly connect to that node to quickly fulfill the request.

The Routable Query API provides two classes for performing routable queries: `VerticaRoutableExecutor` and `VGet`. `VerticaRoutableExecutor` provides a more expressive SQL-based API while `VGet` provides a more structured API for programmatic access.

- The **`VerticaRoutableExecutor`** class allows you to use traditional SQL with a reduced feature set to query data on a single node. Note that the client and server must be at at least Release 7.1 SP1 to use the `VerticaRoutableExecutor`.

For joins, the table must be joined on a key column that exists in each table you are joining, and the tables must be segmented on that key. However, this is not true for unsegmented tables, which can always be joined (since all the data in an unsegmented table is available on all nodes).

- The **`VGet`** class does not use traditional SQL syntax. Instead, it uses a data structure that you build by defining predicates and predicate expressions and outputs and output expressions. This class is ideal for doing Key/Value type lookups on single tables.

The data structure used for querying the table must provide a predicate for each segmented column defined in the projection for the table. You must provide, at a minimum, a predicate with a constant value for each segmented column. For example, an `id` with a value of 12234 if the table is segmented only on the `id` column. You can also specify additional predicates for the other, non-segmented, columns in the table. Predicates act like a SQL *WHERE* clause and multiple predicates/predicate expressions apply together with a SQL AND modifier. Predicates must be defined with a constant value. Predicate expressions can be used to refine the query and can contain any arbitrary SQL expressions (such as less than, greater than, and so on) for any of the non-segmented columns in the table.

Java documentation for all classes and methods in the JDBC Driver is available in the Vertica JDBC Documentation.

Note: The JDBC Routable Query API is read-only and requires JDK 1.6 or greater.

See Also

- [Creating Tables and Projections for use with the Routable Query API](#)
- [Creating a Connection for Routable Queries](#)
- [Defining the Query for Routable Queries Using the `VGet` Class](#)

- [Defining the Query for Routable Queries using the VerticaRoutableExecutor Class](#)
- [Routable Query Performance and Troubleshooting](#)

Creating Tables and Projections for use with the Routable Query API

For routable queries, the client needs to determine the appropriate node to get the data. The client does this by comparing all of the projections available for the table and determining the best projection to use to find the single node that contains data. You must create a projection segmented by the key column(s) on at least one table to take full advantage of the Routable Query API. Other tables which join to this table must either have an unsegmented projection, or a projection segmented as described below.

Note: Tables must be segmented by hash for Routable Queries. See [Hash Segmentation Clause](#). Other segmentation types are not supported.

Creating Tables for use with Routable Queries

To create a table that can be used with the Routable Query API, segment (by hash) the table on a uniformly distributed column. Typically, you segment on a primary key. For faster lookups, sort the projection on the same columns on which you segmented. For example, to create a table that is well suited to Routable Queries:

```
CREATE TABLE users (  
  id INT NOT NULL PRIMARY KEY,  
  username VARCHAR(32),  
  email VARCHAR(64),  
  business_unit VARCHAR(16))  
ORDER BY id  
SEGMENTED BY HASH(id)  
ALL NODES;
```

This table is segmented based on the `id` column (and ordered by `id` to make lookups faster). To build a query for this table using the Routable Query API, you only need to provide a single predicate for the `id` column which returns a single row when queried.

However, if you were to add multiple columns to the segmentation clause, such as this table:

```
CREATE TABLE users2 (  
  id INT NOT NULL PRIMARY KEY,  
  username VARCHAR(32),  
  email VARCHAR(64),  
  business_unit VARCHAR(16))
```

```
ORDER BY id, business_unit  
SEGMENTED BY HASH(id, business_unit)  
ALL NODES;
```

Then you would need to provide two predicates when querying the *users2* table, since the segmentation clause uses both the *id* and the *business_unit* columns. However, if you know both *id* and *business_unit* when you perform the queries, then it is beneficial to segment on both columns, as it makes it easier for the client to determine that this projection is the best projection to use to determine the correct node.

Designing Tables for Single-node JOINS

If you plan to use the `VerticaRoutableExecutor` class and join tables during routable queries, then you must segment all tables being joined by the same segmentation key. Typically this key is a Primary/Foreign key on all the tables being joined. For example, the `customer_key` may be the primary key in a customers dimension table, and the same key is a foreign key in a sales fact table. Projections for a `VerticaRoutableExecutor` query using these tables must be segmented by hash on the customer key in each table.

If you want to join with small dimension tables, such as date dimensions, then it may be appropriate to make those tables unsegmented so that the `date_dimension` data exists on all nodes. It is important to note that when joining unsegmented tables, you still must specify a segmented table in the `createRoutableExecutor()` call.

Verifying Existing Projections for Tables

If you have existing tables that are already segmented by hash (for example, on an ID column), then you can determine what predicates are needed to query the table by using the `select get_table_projections('tableName')` command to view the projections associated with the table. The example table displays the following when `select get_table_projections('users')` is run:

```
Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate] [Stats]  
-----  
public.users_b1 [Segmented: Yes] [Seg Cols: "public.users.id"] [K: 1] [public.users_b0] [Safe: Yes]  
[UptoDate: Yes] [Stats: RowCounts]  
public.users_b0 [Segmented: Yes] [Seg Cols: "public.users.id"] [K: 1] [public.users_b1] [Safe: Yes]  
[UptoDate: Yes] [Stats: RowCounts]
```

Note that for each projection, only the `"public.users.id"` column is specified, meaning you need to provide a predicate for this column when you build your query.

If the table was segmented on multiple columns, for example *id* and *business_unit*, then you would need to provide both columns as predicates to the routable query.

Creating a Connection for Routable Queries

The JDBC Routable Query API provides the `VerticaRoutableConnection` (details are available in the JDBC Documentation) interface to connect to a cluster and allow for Routable Queries. This interface provides advanced routing capabilities beyond those of a normal `VerticaConnection`. The `VerticaRoutableConnection` provides access to the `VerticaRoutableExecutor` and `VGet` classes. See [Defining the Query for Routable Queries using the VerticaRoutableExecutor Class](#) and [Defining the Query for Routable Queries Using the VGet Class](#) respectively.

You enable access to this class by setting the `EnableRoutableQueries` JDBC connection property to `true`.

The `VerticaRoutableConnection` maintains an internal pool of connections and a cache of table metadata that is shared by all `VerticaRoutableExecutor/VGet` objects that are produced by the connection's `createRoutableExecutor()/prepareGet()` method. It is also a fully-fledged JDBC connection on its own and supports all the functionality that a `VerticaConnection` supports. When this connection is closed, all pooled connections managed by this `VerticaRoutableConnection` and all child objects are closed too. The connection pool and metadata is only used by child Routable Query operations.

Example:

You can create the connection using a JDBC `DataSource`:

```
com.vertica.jdbc.DataSource jdbcSettings = new com.vertica.jdbc.DataSource();
jdbcSettings.setDatabase("exampleDB");
jdbcSettings.setHost("v_vmart_node0001.example.com");
jdbcSettings.setUserID("dbadmin");
jdbcSettings.setPassword("password");
jdbcSettings.setEnableRoutableQueries(true);
jdbcSettings.setPort((short) 5433);

VerticaRoutableConnection conn;
conn = (VerticaRoutableConnection)jdbcSettings.getConnection();
```

You can also create the connection using a connection string and the `DriverManager.getConnection()` method:

```
String connectionString = "jdbc:vertica://v_vmart_
node0001.example.com:5433/exampleDB?user=dbadmin&password=&EnableRoutableQueries=true";
VerticaRoutableConnection conn = (VerticaRoutableConnection) DriverManager.getConnection(connectionString);
```

Both methods result in a `conn` connection object that is identical.

Note: Avoid opening many `VerticaRoutableConnection` connections because this connection maintains its own private pool of connections which are not shared with other connections. Instead, your application should use a single connection and issue multiple queries through that connection.

In addition to the `setEnableRoutableQueries` property that the Routable Query API adds to the Vertica JDBC connection class, the API also adds additional properties. The complete list is below.

- `EnableRoutableQueries`: Enables Routable Query lookup capability. Default is false.
- `FailOnMultiNodePlans`: If the plan requires more than one node, and `FailOnMultiNodePlans` is true, then the query fails. If it is set to false then a warning is generated and the query continues. However, latency is greatly increased as the Routable Query must first determine the data is on multiple nodes, then a normal query is run using traditional (all node) execution and execution. Defaults to true. Note that this failure cannot occur on simple calls using only predicates and constant values.
- `MetadataCacheLifetime`: The time in seconds to keep projection metadata. The API caches metadata about the projection used for the query (such as projections). The cache is used on subsequent queries to reduce response time. The default is 300 seconds.
- `MaxPooledConnections`: Cluster-wide maximum number of connections to keep in the `VerticaRoutableConnection`'s internal pool. Default 20.
- `MaxPooledConnectionsPerNode`: Per-node maximum number of connections to keep in the `VerticaRoutableConnection`'s internal pool. Default 5.

Defining the Query for Routable Queries using the `VerticaRoutableExecutor` Class

The `VerticaRoutableExecutor` class is used to access table data directly from a single node. `VerticaRoutableExecutor` directly queries Vertica only on the node that has all the data needed for the query, avoiding the distributed planning and execution costs associated with a normal Vertica execution. You can use `VerticaRoutableExecutor` if you need to join tables or use a group by clause, as these operations are not possible using `VGet`.

When using the `VerticaRoutableExecutor` class, you must follow these rules:

- If joining tables, all tables being joined must be segmented (by hash) on the same set of columns referenced in the join predicate, unless the table being joined is unsegmented.

- When using multiple conditions in WHERE clauses, the WHERE clause must use AND between the conditions. Using OR in the WHERE clause causes the query to degenerate to a multi-node plan. OR, IN list, or range conditions on columns *outside* of the join condition are acceptable if the data exists on the same node.
- You can only execute a single statement per request. "Chained" SQL statements are not permitted.
- Your query may be used in a driver-generated subquery to help determine if the query can be executed on a single node. Therefore, you cannot include the semi-colon at the end of the statement and you cannot include SQL comments (using double-dashes, like so: --), as these would cause the driver-generated query to fail.

You create a `VerticaRoutableExecutor` by calling `createRoutableExecutor(schema, table)`; on a connection object. If `schema` is set to null, then the search path is used to find the table.

VerticaRoutableExecutor Methods

`VerticaRoutableExecutor` has the following methods (more details on the class are available in the JDBC Documentation):

- `execute(query string, [column, value | Map])` - Runs the query. Accepts as input the query to be executed, and either:
 - The column and value when the lookup is being done on just a single value. For example:

```
String column = "customer_key";
Integer value = 1;
ResultSet rs = q.execute(query, column, value)
```

- A Java map of the column names and corresponding values if the lookup is being done on one or more columns. For example: `ResultSet rs = q.execute(query, map);`. The table must have at least one projection segmented by a set of columns exactly matching the columns in the map. Note that each column defined in the map must have only one value. You cannot include more than one value for the same column. For example:

```
Map<String, Object> map = new HashMap<String, Object>();
    map.put("customer_key", 1);
    map.put("another_key", 42);
ResultSet rs = q.execute(query, map);
```

The query being executed uses regular SQL. The SQL used must meet the rules of the `VerticaRoutableExecutor` class. For example, you can add limits and sorts, or use aggregate functions, provided the data exists on a single node.

Important: The JDBC client uses the column/value or map arguments to determine on which node to execute the query. You must make sure that the content of the query uses the same values that you provide in the column/value or map arguments.

Note: The following data types cannot be used as column values. Additionally, if a table is segmented on any columns with the following data types then the table cannot be queried using the Routable Query API:

- interval
- timetz
- timestamptz

Note: The driver does not verify the syntax of the query before it sends the query to the server. If your expression is incorrect, then the query fails.

- `close()` - Closes this `VerticaRoutableExecutor` by releasing resources used by this `VerticaRoutableExecutor`. It does not close the parent JDBC connection to Vertica.
- `getWarnings()` - Retrieves the first warning reported by calls on this `VerticaRoutableExecutor`. Additional warnings are chained and can be accessed with the JDBC `getNextWarning()` method.

Example Query Using `VerticaRoutableExecutor`

The following example details how to use `VerticaRoutableExecutor` to execute a query using both a JOIN clause and an aggregate function with a GROUP BY clause. The example also details how to create both a customer and a sales table, and how to segment the tables so they can be joined using the `VerticaRoutableExecutor` class. This example also uses the `date_dimension` table from the `VMart` schema to illustrate how you can also join data on unsegmented tables.

1. Create a table for customer details, and then create the projections which segment on the `customer_key`.

```
CREATE TABLE customers (customer_key INT, customer_name VARCHAR(128), customer_email VARCHAR(128));

CREATE PROJECTION cust_proj_b0 AS
(SELECT *
 FROM customers) SEGMENTED BY HASH (customer_key) ALL NODES;

CREATE PROJECTION cust_proj_b1 AS
(SELECT *
 FROM customers) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 1;

CREATE PROJECTION cust_proj_b2 AS
(SELECT *
 FROM customers) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 2;

SELECT start_refresh();
```

2. Create a sales table, then create the projections which segment on the customer_key. Since both the customer and sales tables are segmented on the same key, you can join them with the `VerticaRoutableExecutor Routable Query` lookup.

```
CREATE TABLE sales (sale_key INT, customer_key INT, date_key INT, sales_amount FLOAT);

CREATE PROJECTION sales_proj_b0 AS
(SELECT *
 FROM sales) SEGMENTED BY HASH (customer_key) ALL NODES;

CREATE PROJECTION sales_proj_b1 AS
(SELECT *
 FROM sales) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 1;

CREATE PROJECTION sales_proj_b2 AS
(SELECT *
 FROM sales) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 2;

SELECT start_refresh();
```

3. Add some sample data:

```
INSERT INTO customers VALUES (1, 'Fred', 'fred@example.com');
INSERT INTO customers VALUES (2, 'Sue', 'Sue@example.com');
INSERT INTO customers VALUES (3, 'Dave', 'Dave@example.com');
INSERT INTO customers VALUES (4, 'Ann', 'Ann@example.com');
INSERT INTO customers VALUES (5, 'Jamie', 'Jamie@example.com');
COMMIT;

INSERT INTO sales VALUES(1, 1, 1, '100.00');
INSERT INTO sales VALUES(2, 2, 2, '200.00');
```

```
INSERT INTO sales VALUES(3, 3, 3, '300.00');
INSERT INTO sales VALUES(4, 4, 4, '400.00');
INSERT INTO sales VALUES(5, 5, 5, '400.00');
INSERT INTO sales VALUES(6, 1, 15, '500.00');
INSERT INTO sales VALUES(7, 1, 15, '400.00');
INSERT INTO sales VALUES(8, 1, 35, '300.00');
INSERT INTO sales VALUES(9, 1, 35, '200.00');
COMMIT;
```

4. Create an unsegmented projection of the VMart date_dimension table for use in this example. Note you must run `SELECT start_refresh();` to unsegment the existing data:

```
=> CREATE PROJECTION date_dim_unsegment AS
  (SELECT *
   FROM date_dimension) UNSEGMENTED ALL NODES;

=> SELECT start_refresh();
```

Using the customer, sales, and date_dimension data, you can now create a Routable Query lookup that uses joins and a group by to query the customers table and return the total number of purchases per day for a given customer:

```
import java.sql.*;
import java.util.HashMap;
import java.util.Map;
import com.vertica.jdbc.kv.*;

public class verticaKV_doc {
    public static void main(String[] args) {
        com.vertica.jdbc.DataSource jdbcSettings
            = new com.vertica.jdbc.DataSource();
        jdbcSettings.setDatabase("VMart");
        jdbcSettings.setHost("vertica.example.com");
        jdbcSettings.setUserID("dbadmin");
        jdbcSettings.setPassword("password");
        jdbcSettings.setEnableRoutableQueries(true);
        jdbcSettings.setFailOnMultiNodePlans(true);
        jdbcSettings.setPort((short) 5433);
        VerticaRoutableConnection conn;
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("customer_key", 1);

        try {
            conn = (VerticaRoutableConnection)
                jdbcSettings.getConnection();
            String table = "customers";
            VerticaRoutableExecutor q = conn.createRoutableExecutor(null, table);
            String query = "select d.date, SUM(s.sales_amount) as Total ";
            query += " from customers as c ";
            query += " join sales as s ";
            query += " on s.customer_key = c.customer_key ";
            query += " join date_dimension as d ";
            query += " on d.date_key = s.date_key ";
        }
    }
}
```

```
        query += " where c.customer_key = " + map.get("customer_key");
        query += " group by (d.date) order by Total DESC";
        ResultSet rs = q.execute(query, map);
        while(rs.next()) {
            System.out.print("Date: " + rs.getString("date") + ": ");
            System.out.println("Amount: " + rs.getString("Total"));
        }
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

The example code outputs:

```
Date: 2012-01-15: Amount: 900.0
Date: 2012-02-04: Amount: 500.0
Date: 2012-01-01: Amount: 100.0
```

Note that your dates may be different, because the VMart schema randomly generates the dates in the date_dimension table.

Defining the Query for Routable Queries Using the VGet Class

The VGet class (details on the class are available in the JDBC Documentation) is used to access table data directly from a single node when you do not need to join the data or use a group by clause. Like VerticaRoutableExecutor, VGet directly queries Vertica nodes that have the data needed for the query, avoiding the distributed planning and execution costs associated with a normal Vertica execution. However, VGet does not use SQL. Instead, you define predicates and values to perform Key/Value type lookups on a single table. VGet is especially suited to doing key/value-type lookups on single tables.

You create a VGet by calling `prepareGet(schema, table/proj)` on a connection object. `prepareGet()` takes the name of the schema and the name of a table or projection as arguments.

VGet Methods

VGet has the following methods:

- `addPredicate(string, object)` - adds a predicate column and a constant value to the query. You must include a predicate for each column on which the table is segmented. The predicate acts as the "WHERE" clause to the query. Multiple `addPredicate()` method calls are joined by AND modifiers. Note that the VGet retains this value after each call to `execute`. To remove it, use `ClearPredicates()`.

Note: The following data types cannot be used as predicates. Additionally, if a table is segmented on any columns with the following data types then the table cannot be queried using the Routable Query API:

- interval
- timetz
- timestamptz

- `addPredicateExpression(string)` - Accepts arbitrary SQL expressions that operate on the table's columns as input to the query. Predicate expressions and predicates are joined by AND modifiers. You can use segmented columns in predicate expressions, but they must also be specified as a regular predicate with `addPredicate()`. Note that the VGet retains this value after each call to `execute`. To remove it, use `ClearPredicates()`.

Note: The driver does not verify the syntax of the expression before it sends it to the server. If your expression is incorrect then the query fails.

- `addOutputColumn(string)` - Adds a column to be included in the output. By default the query runs as `SELECT *` and you do not need to define any output columns to return the data. If you add output columns then you must add all the columns you want returned. Note that the VGet retains this value after each call to `execute`. To remove it, use `ClearOutputs()`.
- `addOutputExpression(string)` - Accepts arbitrary SQL expressions that operate on the table's columns as output. Note that the VGet retains this value after each call to `execute`. To remove it, use `ClearOutputs()`.

Note: The driver does not verify the syntax of the expression before it sends it to the server. If your expression is incorrect then the query fails.

Note: `addOutputExpression()` is not supported when querying Flex Tables. If you attempt to use `addOutputExpression()` on a Flex Table query, then a `SQLFeatureNotSupportedException` is thrown.

- `addSortColumn(string, SortOrder)` - Adds a sort order to an output column. The output column can be either the one returned by the default query (`SELECT *`) or one of the columns defined in `addOutputColumn` or `addOutputExpress`. You can defined multiple sort columns.

- `setLimit(int)` - Sets a limit on the number of results returned. A limit of 0 is unlimited.
- `clearPredicates()` - Removes predicates that were added by `addPredicate()` and `addPredicateExpression()`.
- `clearOutputs()` - Removes outputs added by `addOutput()` and `addOutputExpression()`.
- `clearSortColumns()` - Removes sort columns previously added by `addSortColumn()`.
- `execute()` - Runs the query. Care must be taken to ensure that the predicate columns exist on the table and projection used by `VGet`, and that the expressions do not require multiple nodes to execute. If an expression is sufficiently complex as to require more than one node to execute, `execute()` throws a `SQLException` if the `FailOnMultiNodePlans` connection property is true.
- `close()` - Closes this `VGet` by releasing resources used by this `VGet`. It does not close the parent JDBC connection to Vertica.
- `getWarnings()` - Retrieves the first warning reported by calls on this `VGet`. Additional warnings are chained and can be accessed with the JDBC `getNextWarning()` method.

You call the `execute()` method to run query. By default, the `VGet` fetches all the columns of all the rows that satisfy the logical AND of all the predicates passed via the `addPredicate()` method. To further customize the get operation use the `addOutputColumn()`, `addOutputExpression()`, `addPredicateExpression()`, `addSortColumn()` and `setLimit()` methods.

Note: `VGet` operations span multiple JDBC connections (and multiple Vertica sessions) and do not honor the parent connection's transaction semantics. If consistency is required across multiple executions, the parent `VerticaRoutableConnection`'s consistent read API can be used to guarantee all operations occur at the same epoch.

`VGet` is thread safe, but all methods are synchronized, so threads that share a `VGet` instance are never run in parallel. For better parallelism, each thread should have its own `VGet` instance. Different `VGet` instances that operate on the same table share pooled connections and metadata in a manner that enables a high degree of parallelism.

Example

You can query the table defined in [Creating Tables and Projections for use with the Routable Query API](#) with the following example code. The table defines an id column that is segmented by hash.

```
import java.sql.*;
import com.vertica.jdbc.kv.*;

public class verticaKV2 {
    public static void main(String[] args) {
        com.vertica.jdbc.DataSource jdbcSettings
            = new com.vertica.jdbc.DataSource();
        jdbcSettings.setDatabase("exampleDB");
        jdbcSettings.setHost("v_vmart_node0001.example.com");
        jdbcSettings.setUserID("dbadmin");
        jdbcSettings.setPassword("password");
        jdbcSettings.setEnableRoutableQueries(true);
        jdbcSettings.setPort((short) 5433);

        VerticaRoutableConnection conn;
        try {
            conn = (VerticaRoutableConnection)
                jdbcSettings.getConnection();
            System.out.println("Connected.");
            VGet get = conn.prepareGet("public", "users");
            get.addPredicate("id", 5);
            ResultSet rs = get.execute();
            rs.next();
            System.out.println("ID: " +
                rs.getString("id"));
            System.out.println("Username: "
                + rs.getString("username"));
            System.out.println("Email: "
                + rs.getString("email"));
            System.out.println("Closing Connection.");
            conn.close();
        } catch (SQLException e) {
            System.out.println("Error! Stacktrace:");
            e.printStackTrace();
        }
    }
}
```

The output:

```
Connected.
ID: 5
Username: userE
Email: usere@example.com
Closing Connection.
```

Routable Query Performance and Troubleshooting

This topic details performance considerations and common issues you might encounter when using the Routable Query API.

Using Resource Pools with Routable Queries

Individual Routable Queries are serviced quickly since they directly access a single node and return only one or a few rows of data. However, by default, Vertica resource pools use an AUTO setting for the `execution_parallelism` parameter. When set to AUTO, the setting is determined by the number of CPU cores available and generally results in multi-threaded execution of queries in the resource pool. It is not efficient to create parallel threads on the server because Routable Query operations return data so quickly and Routable Query operations only use a single thread to find a row. To prevent the server from opening unneeded processing threads, you should create a specific resource pool for Routable Query clients. Consider the following settings for the resource pool you use for Routable Queries:

- Set `execution_parallelism` to 1 to force single-threaded queries. This setting improves Routable Query performance.
- Use CPU affinity to limit the resource pool to a specific CPU or CPU set. The setting ensures that the Routable Queries have resources available to them, but it also prevents Routable Queries from significantly impacting performance on the system for other general queries.
- If you do not set a CPU affinity for the resource pool, consider setting the maximum concurrency value of the resource pool to a setting that ensures good performance for Routable Queries, but does not negatively impact the performance of general queries.

Performance Considerations for Routable Query Connections

Because a `VerticaRoutableConnection` opens an internal pool of connections, it is important to configure `MaxPooledConnections` and `MaxPooledConnectionsPerNode` appropriately for your cluster size and the amount of simultaneous client connections. It is possible to impact normal database connections if you are overloading the cluster with `VerticaRoutableConnections`.

The initial connection to the initiator node discovers all other nodes in the cluster. The internal-pool connections are not opened until a `VerticaRoutableExecutor` or `VGet` query is sent. All `VerticaRoutableExecutors/VGets` in a connection object use connections from the internal pool and are limited by the `MaxPooledConnections` settings. Connections remain open until they are closed so a new connection can be opened elsewhere if the connection limit has been reached.

Troubleshooting Routable Queries

Routable Query issues generally fall into two categories:

- Not providing enough predicates.
- Queries having to span multiple nodes.

Predicate Requirements

You must provide the same number of predicates that correspond to the columns of the table segmented by hash. To determine the segmented columns, run `select get_table_projections('tableName')`. You must provide a predicate for each column displayed in the "Seg Cols" field.

For `VGet`, this means you must literally use `addPredicate()` to add each of the columns. For `VerticaRoutableExecutor`, this means you must provide all of the predicates and values in the map sent to `execute()`.

Multi-node Failures

It is possible to define the correct number of predicates, but still have a failure because multiple nodes contain the data. This failure occurs because the projection's data is not segmented in such a way that the data being queried is contained on a single node. Enable logging for the connection and view the logs to verify the projection being used. If the client is not picking the correct projection, then try to query the projection directly by specifying the projection instead of the table in the create/prepare statement, for example:

- Using `VerticaRoutableExecutor`:

```
conn.createRoutableExecutor(schema, table/projection);
```

- Using `VGet`:

```
conn.prepareGet('schema','table/projection')
```

Additionally, you can use the [EXPLAIN](#) command in `vsql` to help determine if your query can run in single node. `EXPLAIN` can help you understand why the query is being run as single or multi-node.

Pre-Segmenting Data Using `VHash`

The `VHash` class is an implementation of the Vertica hash function for use with JDBC client applications.

Hash segmentation in Vertica allows you to segment a projection based on a built-in hash function. The built-in hash function provides even data distribution across some or all nodes in a cluster, resulting in optimal query execution.

Suppose you have several million rows of values spread across thousands of CSV files. Assume that you already have a table segmented by hash. Before you load the values into your database, you probably want to know to which node a particular value loads. For this reason, using VHash can be particularly helpful, by allowing you to pre-segment your data before loading.

The following example shows the VHash class hashing the first column of a file named "testFile.csv". The name of the first column in this file is *meterId*.

Segment the Data Using VHash

This example demonstrates how you can read the testFile.csv file from the local file system and run a hash function on the meterId column. Using the database metadata from a projection, you can then pre-segment the individual rows in the file based on the hash value of meterId.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.UnsupportedEncodingException;
import java.util.*;
import java.io.IOException;
import java.sql.*;

import com.vertica.jdbc.kv.VHash;

public class VerticaKVDoc {

    final Map<String, FileOutputStream> files;
    final Map<String, List<Long>> nodeToHashList;
    String segmentationMetadata;
    List<String> lines;

    public static void main(String[] args) throws Exception {
        try {
            Class.forName("com.vertica.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Could not find the JDBC driver class.");
            e.printStackTrace();
            return;
        }

        Properties myProp = new Properties();
        myProp.put("user", "username");
        myProp.put("password", "password");

        VerticaKVDoc ex = new VerticaKVDoc();

        // Read in the data from a CSV file.
        ex.readLinesFromFile("C:\\testFile.csv");

        try (Connection conn = DriverManager.getConnection(
            "jdbc:vertica://VerticaHost:portNumber/databaseName", myProp)) {
```

```
// Compute the hashes and create FileOutputStreams.
ex.prepareForHashing(conn);

}

// Write to files.
ex.writeLinesToFiles();
}

public VerticaKVDoc() {
    files = new HashMap<String, FileOutputStream>();
    nodeToHashList = new HashMap<String, List<Long>>();
}

public void prepareForHashing(Connection conn) throws SQLException,
    FileNotFoundException {

    // Send a query to Vertica to return the projection segments.
    try (ResultSet rs = conn.createStatement().executeQuery(
        "SELECT get_projection_segments('public.projectionName')") {
        rs.next();
        segmentationMetadata = rs.getString(1);
    }

    // Initialize the data files.
    try (ResultSet rs = conn.createStatement().executeQuery(
        "SELECT node_name FROM nodes")) {
        while (rs.next()) {
            String node = rs.getString(1);
            files.put(node, new FileOutputStream(node + ".csv"));
        }
    }
}

public void writeLinesToFiles() throws UnsupportedEncodingException,
    IOException {
    for (String line : lines) {

        long hashedValue = VHash.hashLong(getMeterIdFromLine(line));

        // Write the row data to that node's data file.
        String node = VHash.getNodeFor(segmentationMetadata, hashedValue);

        FileOutputStream fos = files.get(node);
        fos.write(line.getBytes("UTF-8"));
    }
}

private long getMeterIdFromLine(String line) {

    // In our file, "meterId" is the name of the first column in the file.
    return Long.parseLong(line.split(",")[0]);
}

public void readLinesFromFile(String filename) throws IOException {
    lines = new ArrayList<String>();
    String line;
    try (BufferedReader reader = new BufferedReader(
        new FileReader(filename))) {
        while ((line = reader.readLine()) != null) {
```

```
        lines.add(line);  
    }  
}  
}
```

Programming ADO.NET Applications

The Vertica driver for ADO.NET allows applications written in C# to read data from, update, and load data into Vertica databases. It provides a data adapter ([Vertica Data Adapter](#)) that facilitates reading data from a database into a data set, and then writing changed data from the data set back to the database. It also provides a data reader (`VerticaDataReader`) for reading data. The driver requires the .NET framework version 3.5+.

For more information about ADO.NET, see:

- [Overview of ADO.NET](#)
- [.NET Framework Developer Guide](#)

Note: All of the examples provided in this section are in C#.

Updating ADO.NET Client Code From Previous Driver Versions

Starting in release 5.1.1, the Vertica client drivers have been updated to improve standards compliance, performance, and reliability. As a result, some Vertica-specific features and past incompatibilities have been eliminated. You must update any client code written for the prior versions of the ADO.NET driver to work with the version 5.1.1 driver and beyond.

Auto Commit Change

- All queries are now Auto Committed. The only exception is that queries run using a Transaction are not committed until the `Commit();` method is called.

Performance Improvements

- Prepared INSERT statements now run significantly faster than in previous driver versions. For the best performance, prepared statements should be executed as part of a transaction.

Namespace Change

- The namespace has changed from `vertica` to `Vertica.Data.VerticaClient`

Connection Properties

- DSN is no longer a valid connection string keyword. You cannot connect to Vertica using ADO.net with a DSN.
- The `RowBufferSize` connection property has been renamed to `ResultBufferSize`.
- Getters on the `VerticaConnection` to get various connection string options (`CacheDirectoryPooling`, `MinPoolSize`, `MaxPoolSize`, `SyncNotification`, `Timeout`, `Enlist`, `UseExtendedTypes`, `Password`, `Pooling`, `MinPoolSize`, `MaxPoolSize`) have been removed.
- There is no longer a locale connection string keyword and you cannot set the locale through the connection string. To change the locale, run the query "set locale to..."
- The connection property to enable or disable auto commit has been removed. All queries outside of transactions are auto-committed.

Result Buffering

- The driver now buffers all results, and always uses streaming. Because of this, the following functionality has changed:
 - `VerticaCommandBehavior` enum has been removed. This enum extended the ADO.NET `CommandBehavior` enum to add support for buffering results. Results are now buffered in Vertica 5.1.x.
 - The `VerticaCommand.ExecuteReader(CommandBehavior, bool)` argument has been removed.
 - `CacheDirectory` or `PreloadReader` connection string options have been removed.

Logging Changes

- Log properties are no longer configured on the connection string. Log properties are now configured through the [VerticaLogProperties](#) class.

Data Type Changes

The following data types have changed:

Old Datatype Name	New Datatype Name
<code>verticaType.Integer</code>	<code>VerticaType.BigInt</code>
<code>verticaType.Bigint</code>	<code>VerticaType.BigInt</code>
<code>verticaType.Timestamp</code>	<code>VerticaType.DateTime</code>
<code>verticaType.Interval</code>	Changed to specific type of interval, for example: <ul style="list-style-type: none">• <code>VerticaType.IntervalDay</code>• <code>VerticaType.IntervalDayToHour</code>• <code>VerticaType.IntervalDayToMinute</code>• etc.
<code>verticaType.Real</code>	<code>VerticaType.Double</code>
<code>verticaType.Text</code>	<code>VerticaType.VarChar</code>
<code>verticaType.Smallint</code>	<code>VerticaType.BigInt</code>
<code>verticaType.Varbinary</code>	<code>VerticaType.VarBinary</code>

Multiple Commands Now Supported

Multiple commands in a single statement are now supported, provided that parameters are not used in any of the commands in the statement. The exception is COPY commands. You cannot issue multiple COPY commands in the same statement.

Setting the Locale for ADO.NET Sessions

- ADO.NET applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. The same cautions as for ODBC apply if this encoding is violated.
- The ADO.NET driver converts UTF-16 data to UTF-8 when passing to the Vertica server and converts data sent by Vertica server from UTF-8 to UTF-16
- ADO.NET applications should set the correct server session locale by executing the [SET LOCALE TO](#) command in order to get expected collation and string functions behavior on the server.
- If there is no default session locale at the database level, ADO.NET applications need to set the correct server session locale by executing the [SET LOCALE TO](#) command in order to get expected collation and string functions behavior on the server. See the [SET LOCALE](#) command in the SQL Reference Manual

Connecting to the Database

This section describes:

- [Using SSL: Installing SSL Certificates on Windows](#)
- [Opening and Closing the Database Connection \(ADO.NET\)](#)
- [ADO.NET Connection Properties](#)
- [Configuring Log Properties](#)

Using SSL: Installing Certificates on Windows

You can optionally secure communication between your ADO.NET application and Vertica using SSL. The Vertica ADO.NET driver uses the default Windows key store when looking for SSL certificates. This is the same key store that Internet Explorer uses.

Before you can use SSL on the client side, you must implement SSL on the server. See [TLS/SSL Server Authentication](#) in the Administrator's Guide, perform those steps, then return to this topic to install the SSL certificate on Windows.

To use SSL for ADO.NET connections to Vertica:

- Import the server and client certificates into the Windows Key Store.
- If required by your certificates, import the public certificate of your Certifying Authority.

Import the Server and Client Certificates into the Windows Key store:

1. Copy the server.crt file you generated when you [enabled SSL](#) on the server to your Windows Machine.
2. Double-click the certificate.
3. Let Windows determine the key type, and click **Install**.

Import the Public Certificate of Your CA:

You must establish a chain of trust for the certificates. You may need to import the public certificate for your Certifying Authority (CA) (especially if it is a self-signed certificate).

1. using the same certificate as above, double-click the certificate.
2. Select **Place all certificates in the following store**.
3. Click **Browse**, select **Trusted Root Certification Authorities** and click **Next**.
4. Click **Install**.

Enable SSL in Your ADO.NET Applications

In your connection string, be sure to enable SSL by setting the SSL property in `VerticaConnectionStringBuilder` to true, for example:

```
//configure connection properties      VerticaConnectionStringBuilder builder = new
VerticaConnectionStringBuilder();
    builder.Host = "192.168.17.10";
    builder.Database = "VMart";
    builder.User = "dbadmin";
    builder.SSL = true;
    //open the connection
    VerticaConnection _conn = new VerticaConnection(builder.ToString());
    _conn.Open();
```

Opening and Closing the Database Connection (ADO.NET)

Before you can access data in Vertica through ADO.NET, you must create a connection to the database using the `VerticaConnection` class which is an implementation of `System.Data.DbConnection`. The `VerticaConnection` class takes a single argument that contains the connection properties as a string. You can manually create a string of property keywords to use as the argument, or you can use the `VerticaConnectionStringBuilder` class to build a connection string for you.

This topic details the following:

- Manually building a connection string and connecting to Vertica
- Using `VerticaConnectionStringBuilder` to create the connection string and connecting to Vertica
- Closing the connection

To Manually Create a Connection string:

See [ADO.NET Connection Properties](#) for a list of available properties to use in your connection string. At a minimum, you need to specify the Host, Database, and User.

1. For each property, provide a value and append the properties and values one after the other, separated by a semicolon. Assign this string to a variable. For example:

```
String connectString = "DATABASE=VMart;HOST=v_vmart_node0001;USER=dbadmin";
```

2. Build a Vertica connection object that specifies your connection string.

```
VerticaConnection _conn = new VerticaConnection(connectString)
```

3. Open the connection.

```
_conn.Open();
```

4. Create a command object and associate it with a connection. All `VerticaCommand` objects must be associated with a connection.

```
VerticaCommand command = _conn.CreateCommand();
```

To Use the `VerticaConnectionStringBuilder` Class to Create a Connection String and Open a connection:

1. Create a new object of the `VerticaConnectionStringBuilder` class.

```
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
```

2. Update your `VerticaConnectionStringBuilder` object with property values. See [ADO.NET Connection Properties](#) for a list of available properties to use in your connection string. At a minimum, you need to specify the Host, Database, and User.

```
builder.Host = "v_vmart_node0001";  
builder.Database = "VMart";  
builder.User = "dbadmin";
```

3. Build a Vertica connection object that specifies your connection `VerticaConnectionStringBuilder` object as a string.

```
VerticaConnection _conn = new VerticaConnection(builder.ToString());
```

4. Open the connection.

```
_conn.Open();
```

5. Create a command object and associate it with a connection. All `VerticaCommand` objects must be associated with a connection.

```
VerticaCommand command = _conn.CreateCommand;
```

Note: If your database is not in compliance with your Vertica license, the call to `VerticaConnection.open()` returns a warning message to the console and the log. See [Managing Licenses](#) in the Administrator's Guide for more information.

To Close the connection:

When you're finished with the database, close the connection. Failure to close the connection can deteriorate the performance and scalability of your application. It can also prevent other clients from obtaining locks.

```
_conn.Close();
```

Example Usage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            //Perform some operations
            _conn.Close();
        }
    }
}
```

ADO.NET Connection Properties

You use connection properties to configure the connection between your ADO.NET client application and your Vertica database. The properties provide the basic information about the connections, such as the server name and port number, needed to connect to your database.

You can set a connection property in two ways:

- Include the property name and value as part of the connection string you pass to a `VerticaConnection`.
- Set the properties in a `VerticaConnectionStringBuilder` object, and then pass the object as a string to a `VerticaConnection`.

Property	Description	Default Value
Database	Name of the Vertica database to which you want to connect. For example, if you installed the example VMart database, the database is "VMart".	none

Property	Description	Default Value
User	Name of the user to log into Vertica.	none
Port	Port on which Vertica is running.	5433
Host	<p>The host name or IP address of the server on which Vertica is running.</p> <p>You can provide an IPv4 address, IPv6 address, or host name.</p> <p>In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.</p>	none
PreferredAddressFamily	<p>The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:</p> <ul style="list-style-type: none"> • <code>Ipv4</code>—Connect to the server using IPv4. • <code>Ipv6</code>—Connect to the server using IPv6. • <code>None</code>—Use the IP address provided by the DNS server. 	Vertica.Data.VerticaClient.AddressFamilyP reference.None
Password	The password associated with the user connecting to the server.	string.Empty

Property	Description	Default Value
BinaryTransfer	<p>Provides a Boolean value that, when set to true, uses binary transfer instead of string transfer. When set to false, the ADO.NET connection uses string transfer. Binary transfer provides faster performance in reading data from a server to an ADO.NET client. Binary transfer also requires less bandwidth than string transfer, although it sometimes uses more when transferring a large number of small values.</p>	true
ConnSettings	<p>SQL commands to run upon connection. Uses %3B for semicolons.</p>	string.Empty
IsolationLevel	<p>Sets the transaction isolation level for Vertica. See Transactions for a description of the different transaction levels. This value is either Serializable, ReadCommitted, or Unspecified. See Setting the Transaction Isolation Level for an example of setting the isolation level using this keyword.</p> <p>Note: By default, this value is set to IsolationLevel.Unspecified, which means the connection uses the server's default transaction isolation level. Vertica's default isolation level is</p>	System.Data. IsolationLevel.Unspecified

Property	Description	Default Value
	IsolationLevel.ReadCommitted.	
Label	A string to identify the session on the server.	string
DirectBatchInsert	A Boolean value, whether to bulk insert to ROS (true) or WOS (false).	false
ResultBufferSize	The size of the buffer to use when streaming results. A value of 0 means ResultBufferSize is turned off.	8192
ConnectionTimeout	Number seconds to wait for a connection. A value of 0 means no timeout.	0
ReadOnly	A Boolean value. If true, throw an exception on write attempts.	false
Pooling	A boolean value, whether to enable connection pooling. Connection pooling is useful for server applications because it allows the server to reuse connections. This saves resources and enhances the performance of executing commands on the database. It also reduces the amount of time a user must wait to establish a connection to the database	false
MinPoolSize	An integer that defines the minimum number of connections to pool.	1

Property	Description	Default Value
	<p>Valid Values: Cannot be greater than the number of connections that the server is configured to allow. Otherwise, an exception results.</p> <p>Default: 55</p>	
MaxPoolSize	<p>An integer that defines the maximum number of connections to pool.</p> <p>Valid Values: Cannot be greater than the number of connections that the server is configured to allow. Otherwise, an exception results.</p>	20
LoadBalanceTimeout	<p>The amount of time, expressed in seconds, to timeout or remove unused pooled connections.</p> <p>Disable: Set to 0 (no timeouts)</p> <p>If you are using a cluster environment to load-balance the work, then pool is restricted to the servers in the cluster when the pool was created. If additional servers are added to the cluster, and the pool is not removed, then the new servers are never added to the connection pool unless LoadBalanceTimeout is set and exceeded or <code>VerticaConnection.Clea</code></p>	0 (no timeout)

Property	Description	Default Value
	rAllPools() is called manually from an application. If you are using load balancing, then set this property to a value that considers when new servers are added to the cluster. However, do not set it so low that pools are frequently removed and rebuilt, doing so makes pooling ineffective.	
SSL	A Boolean value, indicating whether to use SSL for the connection.	false
IntegratedSecurity	Provides a Boolean value that, when set to true, uses the user's Windows credentials for authentication, instead of user/password in the connection string.	false
KerberosServiceName	Provides the service name portion of the Vertica Kerberos principal; for example: vertica/host@EXAMPLE.COM	vertica
KerberosHostname	Provides the instance or host name portion of the Vertica Kerberos principal; for example: vertica/ host@EXAMPLE.COM	Value specified in the servername connection string property

Enabling Native Connection Load Balancing in ADO.NET

Native connection load balancing helps spread the overhead caused by client connections on the hosts in the Vertica database. Both the server and the client must enable native connection load balancing in order for it to have an effect. If both have enabled it, then when the client initially connects to a host in the database, the host picks a host to handle the client connection from a list of the currently up hosts in the database, and informs the client which host it has chosen. If the initially-contacted host did not choose itself to handle the connection, the client disconnects, then opens a second connection to the host selected by the first host. The connection process to this second host proceeds as usual—if SSL is enabled, then SSL negotiations begin, otherwise the client begins the authentication process. See [About Native Connection Load Balancing](#) in the Administrator's Guide for details.

To enable native load balancing on your client, set the `ConnectionLoadBalance` connection parameter to `true` either in the connection string or using the `ConnectionStringBuilder` (`ConnectionStringBuilder`). The following example demonstrates connecting to the database several times with native connection load balancing enabled, and fetching the name of the node handling the connection from the `V_MONITOR.CURRENT_SESSION` system table.

```
using System;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder =
                new VerticaConnectionStringBuilder();
            builder.Host = "v_vmart_node0001.example.com";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            // Enable native client load balancing in the client,
            // must also be enabled on the server!
            builder.ConnectionLoadBalance = true;
            // Connect 3 times to verify a new node is connected
            // for each connection.
            for (int i = 1; i <= 4; i++)
            {
                try
                {
                    VerticaConnection _conn =
                        new VerticaConnection(builder.ToString());
                    _conn.Open();
                    if (i == 1)
                    {
                        // On the first connection, check the server policy for load balance
```

```
        VerticaCommand sqlcom = _conn.CreateCommand();
        sqlcom.CommandText =
"SELECT LOAD_BALANCE_POLICY FROM V_CATALOG.DATABASES";
        var returnValue = sqlcom.ExecuteScalar();
        Console.WriteLine("Status of load balancy policy
on server: " + returnValue.ToString() + "\n");
    }
    VerticaCommand command = _conn.CreateCommand();
    command.CommandText =
"SELECT node_name FROM V_MONITOR.CURRENT_SESSION";
    VerticaDataReader dr = command.ExecuteReader();
    while (dr.Read())
    {
        Console.Write("Connect attempt #" + i + "... ");
        Console.WriteLine("Connected to node " + dr[0]);
    }
    dr.Close();
    _conn.Close();
    Console.WriteLine("Disconnecting.\n");
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}
}
}
}
```

Running the above example produces the following output:

```
Status of load balancy policy on server: roundrobin

Connect attempt #1... Connected to node v_vmart_node0001
Disconnecting.

Connect attempt #2... Connected to node v_vmart_node0002
Disconnecting.

Connect attempt #3... Connected to node v_vmart_node0003
Disconnecting.

Connect attempt #4... Connected to node v_vmart_node0001
Disconnecting.
```

ADO.NET Connection Failover

If a client application attempts to connect to a host in the Vertica Analytics Platform cluster that is down, the connection attempt fails when using the default connection configuration. This failure usually returns an error to the user. The user must either wait until the host recovers and retry the connection or manually edit the connection settings to choose another host.

Due to Vertica Analytics Platform's distributed architecture, you usually do not care which database host handles a client application's connection. You can use the client driver's connection failover feature to prevent the user from getting connection errors when the host specified in the connection settings is unreachable. It gives you two ways to let the client driver automatically attempt to connect to a different host if the one specified in the connection parameters is unreachable:

- Configure your DNS server to return multiple IP addresses for a host name. When you use this host name in the connection settings, the client attempts to connect to the first IP address from the DNS lookup. If the host at that IP address is unreachable, the client tries to connect to the second IP, and so on until it either manages to connect to a host or it runs out of IP addresses.
- Supply a list of backup hosts for the client driver to try if the primary host you specify in the connection parameters is unreachable.

For both methods, the process of failover is transparent to the client application (other than specifying the list of backup hosts, if you choose to use the list method of failover). If the primary host is unreachable, the client driver automatically tries to connect to other hosts.

Failover only applies to the initial establishment of the client connection. If the connection breaks, the driver does not automatically try to reconnect to another host in the database.

Choosing a Failover Method

You usually choose to use one of the two failover methods. However, they do work together. If your DNS server returns multiple IP addresses and you supply a list of backup hosts, the client first tries all of the IPs returned by the DNS server, then the hosts in the backup list.

Note: If a host name in the backup host list resolves to multiple IP addresses, the client does not try all of them. It just tries the first IP address in the list.

The DNS method of failover centralizes the configuration client failover. As you add new nodes to your Vertica Analytics Platform cluster, you can choose to add them to the failover list by editing the DNS server settings. All client systems that use the DNS server to connect to Vertica Analytics Platform automatically use connection failover without having to change any settings. However, this method does require administrative access to the DNS server that all clients use to connect to the Vertica Analytics Platform cluster. This may not be possible in your organization.

Using the backup server list is easier than editing the DNS server settings. However, it decentralizes the failover feature. You may need to update the application settings on each client system if you make changes to your Vertica Analytics Platform cluster.

Using DNS Failover

To use DNS failover, you need to change your DNS server's settings to map a single host name to multiple IP addresses of hosts in your Vertica Analytics Platform cluster. You then have all client applications use this host name to connect to Vertica Analytics Platform.

You can choose to have your DNS server return as many IP addresses for the host name as you want. In smaller clusters, you may choose to have it return the IP addresses of all of the hosts in your cluster. However, for larger clusters, you should consider choosing a subset of the hosts to return. Otherwise there can be a long delay as the client driver tries unsuccessfully to connect to each host in a database that is down.

Using the Backup Host List

To enable backup list-based connection failover, your client application has to specify at least one IP address or host name of a host in the `BackupServerNode` parameter. The host name or IP can optionally be followed by a colon and a port number. If not supplied, the driver defaults to the standard Vertica port number (5433). To list multiple hosts, separate them by a comma.

The following example demonstrates setting the `BackupServerNode` connection parameter to specify additional hosts for the connection attempt. The connection string intentionally has a non-existent node, so that the initial connection fails. The client driver has to resort to trying the backup hosts to establish a connection to Vertica.

```
using System;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder =
                new VerticaConnectionStringBuilder();
            builder.Host = "not.a.real.host:5433";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            builder.BackupServerNode =
                "another.broken.node:5433,v_vmart_node0002.example.com:5433";
            try
            {
                VerticaConnection _conn =
                    new VerticaConnection(builder.ToString());
                _conn.Open();
            }
        }
    }
}
```

```
VerticaCommand sqlcom = _conn.CreateCommand();
sqlcom.CommandText = "SELECT node_name FROM current_session";
var returnValue = sqlcom.ExecuteScalar();
Console.WriteLine("Connected to node: " +
    returnValue.ToString() + "\n");
_conn.Close();
Console.WriteLine("Disconnecting.\n");
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}
}
```

Notes

- When native connection load balancing is enabled, the additional servers specified in the BackupServerNode connection parameter are only used for the initial connection to a Vertica host. If host redirects the client to another host in the database cluster to handle its connection request, the second connection does not use the backup node list. This is rarely an issue, since native connection load balancing is aware of which nodes are currently up in the database. See [Enabling Native Connection Load Balancing in ADO.NET](#).
- Connections to a host taken from the BackupServerNode list are not pooled for ADO.NET connections.

Configuring Log Properties (ADO.Net)

Log properties for ADO.Net are configured differently than they are other client drivers. On the other client drivers, log properties can be configured as one of the connection properties. The ADO.Net driver user the VerticaLogProperties class to configure the properties.

VerticaLogProperties

VerticaLogProperties is a static class that allows you to set and get the log settings for the ADO.net driver. You can control the log level, log path, and log namespace using this class.

The log is created when the first connection is opened. Once the connection is opened, you cannot change the log path. It must be set prior to opening the connection. You can change the log level and log namespace at any time.

Setting Log Properties

Setting the log properties is done using the three methods in the `VerticaLogProperties` class. The three methods are:

- `SetLogPath(String path, bool persist)`
- `SetLogNamespace(String lognamespace, bool persist)`
- `SetLogLevel(VerticaLogLevel loglevel, bool persist)`

Each of the methods requires a boolean `persist` argument. When set to true, the `persist` argument causes the setting to be written to the client's Windows Registry, where it is used for all subsequent connections. If set to false, then the log property only applies to the current session.

SetLogPath

The `SetLogPath` method takes as its arguments a string containing the path to the log file and the `persist` argument. If the path string contains only a directory path, then the log file is created with the name `vdp-driver-MM-dd_HH.mm.ss.log` (where `MM-dd_HH.mm.ss` is the date and time the log was created). If the path ends in a filename, such as `log.txt` or `log.log`, then the log is created with that filename.

If `SetLogPath` is called with an empty string for the path argument, then the client executable's current directory is used as the log path.

If `SetLogPath` is not called and no registry entry exists for the log path, and you have called any of the other `VerticaLogProperties` methods, then the client executable's current directory is used as the log path.

When the `persist` argument is set to true, the path specified is copied to the registry verbatim. If no filename was specified, then the filename is not saved to the registry.

Note: Note: The path must exist on the client system prior to calling this method. The method does not create directories.

Example Usage:

```
//set the log path
string path = "C:\\log";
VerticaLogProperties.SetLogPath(path, false);
```

SetLogNamespace

The `SetLogNamespace` method takes as its arguments a string containing the namespace to log and the `persist` argument. The namespace string to log can be one of the following:

- `Vertica`
- `Vertica.Data.VerticaClient`
- `Vertica.Data.Internal.IO`
- `Vertica.Data.Internal.DataEngine`
- `Vertica.Data.Internal.Core`

Namespaces can be truncated to include multiple child namespaces. For example, you can specify `"Vertica.Data.Internal"` to log for all of the `Vertica.Data.Internal` namespaces.

If a log namespace is not set, and no value is stored in the registry, then the `"Vertica"` namespace is used for logging.

Example Usage:

```
//set namespace to log
string lognamespace = "Vertica.Data.VerticaClient";
VerticaLogProperties.SetLogNamespace(lognamespace, false);
```

SetLogLevel

The `SetLogLevel` method takes as its arguments a `VerticaLogLevel` type and the `persist` argument. The `VerticaLogLevel` argument can be one of:

- `VerticaLogLevel.None`
- `VerticaLogLevel.Fatal`
- `VerticaLogLevel.Error`
- `VerticaLogLevel.Warning`
- `VerticaLogLevel.Info`
- `VerticaLogLevel.Debug`
- `VerticaLogLevel.Trace`

If a log level is not set, and no value is stored in the registry, then `VerticaLogLevel.None` is used.

Example Usage:

```
//set log level
VerticaLogLevel level = VerticaLogLevel.Debug;
VerticaLogProperties.SetLogLevel(level, false);
```

Getting Log Properties

You can get the log property values using the getters included in the `VerticaLogProperties` class. The properties are:

- `LogPath`
- `LogNamespace`
- `LogLevel`

Example Usage:

```
//get current log settings
string logpath = VerticaLogProperties.LogPath;
VerticaLogLevel loglevel = VerticaLogProperties.LogLevel;
string logns = VerticaLogProperties.LogNamespace;
Console.WriteLine("Current Log Settings:");
Console.WriteLine("Log Path: " + logpath);
Console.WriteLine("Log Level: " + loglevel);
Console.WriteLine("Log Namespace: " + logns);
```

Setting and Getting Log Properties Example

This complete example shows how to set and get log properties:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
//configure connection properties
```

```
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();  
builder.Host = "192.168.1.10";  
builder.Database = "VMart";  
builder.User = "dbadmin";
```

//get current log settings

```
string logpath = VerticaLogProperties.LogPath;  
VerticaLogLevel loglevel = VerticaLogProperties.LogLevel;  
string logns = VerticaLogProperties.LogNamespace;  
Console.WriteLine("\nOld Log Settings:");  
Console.WriteLine("Log Path: " + logpath);  
Console.WriteLine("Log Level: " + loglevel);  
Console.WriteLine("Log Namespace: " + logns);
```

//set the log path

```
string path = "C:\\log";  
VerticaLogProperties.SetLogPath(path, false);
```

//set log level

```
VerticaLogLevel level = VerticaLogLevel.Debug;  
VerticaLogProperties.SetLogLevel(level, false);
```

//set namespace to log

```
string lognamespace = "Vertica";  
VerticaLogProperties.SetLogNamespace(lognamespace, false);
```

//open the connection

```
VerticaConnection _conn = new VerticaConnection(builder.ToString());  
_conn.Open();
```

//get new log settings

```
logpath = VerticaLogProperties.LogPath;  
loglevel = VerticaLogProperties.LogLevel;  
logns = VerticaLogProperties.LogNamespace;  
Console.WriteLine("\nNew Log Settings:");  
Console.WriteLine("Log Path: " + logpath);  
Console.WriteLine("Log Level: " + loglevel);  
Console.WriteLine("Log Namespace: " + logns);
```

//close the connection

```
        _conn.Close();    }  
    }  
}
```

The example produces the following output:

```
Old Log Settings:  
Log Path:  
Log Level: None  
Log Namespace:  
New Log Settings:  
Log Path: C:\log  
Log Level: Debug  
Log Namespace: Vertica
```

Querying the Database Using ADO.NET

This section describes how to create queries to do the following:

- [Inserting data into the database](#)
- [Read data from the database](#)
- [Load data into the database](#)

Note: The `ExecuteNonQuery()` method used to query the database returns an `int32` with the number of rows affected by the query. The maximum size of an `int32` type is a constant and is defined to be 2,147,483,547. If your query returns more results than the `int32` max, then ADO.NET throws an exception because of the overflow of the `int32` type. However the query is still processed by Vertica even when the reporting of the return value fails. This is a limitation in .NET, as `ExecuteNonQuery()` is part of the standard ADO.NET interface.

Inserting Data (ADO.NET)

Inserting data can be done using the `VerticaCommand` class. `VerticaCommand` is an implementation of `DbCommand`. It allows you to create and send a SQL statement to the database. Use the `CommandText` method to assign a SQL statement to the command and then execute the SQL by calling the `ExecuteNonQuery` method. The `ExecuteNonQuery` method is used for executing statements that do not return result sets.

To Insert a Single Row of data:

1. [Create a connection to the database.](#)
2. Create a command object using the connection.

```
VerticaCommand command = _conn.CreateCommand();
```

3. Insert data using an INSERT statement. The following is an example of a simple insert. Note that it does not contain a COMMIT statement because the Vertica ADO.NET driver operates in autocommit mode.

```
command.CommandText =  
    "INSERT into test values(2, 'username', 'email', 'password');"
```

4. Execute the query. The rowsAdded variable contains the number of rows added by the insert statement.

```
Int32 rowsAdded = command.ExecuteNonQuery();
```

The ExecuteNonQuery() method returns the number of rows affected by the command for UPDATE, INSERT, and DELETE statements. For all other types of statements it returns -1. If a rollback occurs then it is also set to -1.

Example Usage:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Data;  
using Vertica.Data.VerticaClient;  
namespace ConsoleApplication  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();  
            builder.Host = "192.168.1.10";  
            builder.Database = "VMart";  
            builder.User = "dbadmin";  
            VerticaConnection _conn = new VerticaConnection(builder.ToString());  
            _conn.Open();  
            VerticaCommand command = _conn.CreateCommand();  
            command.CommandText =  
                "INSERT into test values(2, 'username', 'email', 'password');"  
            Int32 rowsAdded = command.ExecuteNonQuery();  
            Console.WriteLine( rowsAdded + " rows added!");  
            _conn.Close();  
        }  
    }  
}
```

Using Parameters

You can use parameters to execute similar SQL statements repeatedly and efficiently.

Using Parameters

VerticaParameters are an extension of the System.Data.DbParameter base class in ADO.NET and are used to set parameters in commands sent to the server. Use Parameters in all queries (SELECT/INSERT/UPDATE/DELETE) for which the values in the WHERE clause are not static; that is for all queries that have a known set of columns, but whose filter criteria is set dynamically by an application or end user. Using parameters in this way greatly decreases the chances of a SQL injection issue that can occur when simply creating a SQL query from a number of variables.

Parameters require that a valid DbType, VerticaDbType, or System type be assigned to the parameter. See [SQL Data Types](#) and [ADO.NET Data Types](#) for a mapping of System, Vertica, and DbTypes.

To create a parameter placeholder, place either the at sign (@) or a colon (:) character in front of the parameter name in the actual query string. Do not insert any spaces between the placeholder indicator (@ or :) and the placeholder.

Note: The @ character is the preferred way to identify parameters. The colon (:) character is supported for backward compatibility.

For example, the following typical query uses the string 'MA' as a filter.

```
SELECT customer_name, customer_address, customer_city, customer_state
FROM customer_dimension WHERE customer_state = 'MA';
```

Instead, the query can be written to use a parameter. In the following example, the string MA is replaced by the parameter placeholder @STATE.

```
SELECT customer_name, customer_address, customer_city, customer_state
FROM customer_dimension WHERE customer_state = @STATE;
```

For example, the ADO.net code for the prior example would be written as:

```
VerticaCommand command = _conn.CreateCommand();
command.CommandText = "SELECT customer_name, customer_address, customer_city, customer_state
    FROM customer_dimension WHERE customer_state = @STATE";
command.Parameters.Add(new VerticaParameter( "STATE", VerticaType.VarChar));
command.Parameters["STATE"].Value = "MA";
```

Note: Although the VerticaCommand class supports a Prepare() method, you do not need to call the Prepare() method for parameterized statements because Vertica automatically prepares the statement for you.

Creating and Rolling Back Transactions

Creating Transactions

Transactions in Vertica are atomic, consistent, isolated, and durable. When you connect to a database using the Vertica ADO.NET Driver, the connection is in autocommit mode and each individual query is committed upon execution. You can collect multiple statements into a single transaction and commit them at the same time by using a transaction. You can also choose to rollback a transaction before it is committed if your code determines that a transaction should not commit.

Transactions use the `VerticaTransaction` object, which is an implementation of `DbTransaction`. You must associate the transaction with the `VerticaCommand` object.

The following code uses an explicit transaction to insert one row each into two tables of the `VMart` schema.

To Create a Transaction in Vertica Using the ADO.NET driver:

1. [Create a connection to the database.](#)
2. Create a command object using the connection.

```
VerticaCommand command = _conn.CreateCommand();
```

3. Start an explicit transaction, and associate the command with it.

```
VerticaTransaction txn = _conn.BeginTransaction();  
command.Connection = _conn;  
command.Transaction = txn;
```

4. Execute the individual SQL statements to add rows.

```
command.CommandText =  
    "insert into product_dimension values( ... )";  
command.ExecuteNonQuery();  
command.CommandText =  
    "insert into store_orders_fact values( ... )";
```

5. Commit the transaction.

```
txn.Commit();
```

Rolling Back Transactions

If your code checks for errors, then you can catch the error and rollback the entire transaction.

```
VerticaTransaction txn = _conn.BeginTransaction();
VerticaCommand command = new
    VerticaCommand("insert into product_dimension values( 838929, 5, 'New item 5' )", _conn);
// execute the insert
command.ExecuteNonQuery();
command.CommandText = "insert into product_dimension values( 838929, 6, 'New item 6' )";
// try insert and catch any errors
bool error = false;
try
{
    command.ExecuteNonQuery();
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    error = true;
}
if (error)
{
    txn.Rollback();
    Console.WriteLine("Errors. Rolling Back.");
}
else
{
    txn.Commit();
    Console.WriteLine("Queries Successful. Committing.");
}
```

Commit and Rollback Example

This example details how you can commit or rollback queries during a transaction.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            bool error = false;
            VerticaCommand command = _conn.CreateCommand();
            VerticaCommand command2 = _conn.CreateCommand();
            VerticaTransaction txn = _conn.BeginTransaction();
            command.Connection = _conn;
            command.Transaction = txn;
            command.CommandText =
                "insert into test values(1, 'test', 'test', 'test' )";
            Console.WriteLine(command.CommandText);
            try
            {
                command.ExecuteNonQuery();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                error = true;
            }
            command.CommandText =
                "insert into test values(2, 'ear', 'eye', 'nose', 'extra' )";
            Console.WriteLine(command.CommandText);
            try
            {
                command.ExecuteNonQuery();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                error = true;
            }
            if (error)
            {
                txn.Rollback();
                Console.WriteLine("Errors. Rolling Back.");
            }
        }
    }
}
```

```
        else
        {
            txn.Commit();
            Console.WriteLine("Queries Successful. Committing.");
        }
        _conn.Close();
    }
}
```

The example displays the following output on the console:

```
insert into test values(1, 'test', 'test', 'test' )
insert into test values(2, 'ear', 'eye', 'nose', 'extra' )
[42601]ERROR: INSERT has more expressions than target columns
Errors. Rolling Back.
```

See Also

- [Setting the Transaction Isolation Level](#)

Setting the Transaction Isolation Level

You can set the transaction isolation level on a per-connection and per-transaction basis. See [Transaction](#) for an overview of the transaction isolation levels supported in Vertica. To set the default transaction isolation level for a connection, use the *IsolationLevel* keyword in the `VerticaConnectionStringBuilder` string (see [Connection String Keywords](#) for details). To set the isolation level for an individual transaction, pass the isolation level to the `VerticaConnection.BeginTransaction()` method call to start the transaction.

To set the Isolation Level on a connection-basis:

1. Use the `VerticaConnectionStringBuilder` to build the connection string.
2. Provide a value for the `IsolationLevel` builder string. It can take one of two values: `IsolationLevel.ReadCommitted` (default) or `IsolationLevel.Serializable`. For example:

```
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
builder.Host = "192.168.1.100";
builder.Database = "VMart";
builder.User = "dbadmin";
builder.IsolationLevel = System.Data.IsolationLevel.Serializable
VerticaConnection _conn1 = new VerticaConnection(builder.ToString());
_conn1.Open();
```

To set the Isolation Level on a Transaction basis:

1. Set the IsolationLevel on the BeginTransaction method, for example

```
VerticaTransaction txn = _conn.BeginTransaction(IsolationLevel.Serializable);
```

Example usage:

The following example demonstrates:

- getting the connection's transaction isolation level.
- setting the connection's isolation level using connection property.
- setting the transaction isolation level for a new transaction.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn1 = new VerticaConnection(builder.ToString());
            _conn1.Open();
            VerticaTransaction txn1 = _conn1.BeginTransaction();
            Console.WriteLine("\n Transaction 1 Transaction Isolation Level: " +
                txn1.IsolationLevel.ToString());
            txn1.Rollback();
            VerticaTransaction txn2 = _conn1.BeginTransaction(IsolationLevel.Serializable);
            Console.WriteLine("\n Transaction 2 Transaction Isolation Level: " +
                txn2.IsolationLevel.ToString());
            txn2.Rollback();
            VerticaTransaction txn3 = _conn1.BeginTransaction(IsolationLevel.ReadCommitted);
            Console.WriteLine("\n Transaction 3 Transaction Isolation Level: " +
                txn3.IsolationLevel.ToString());
            _conn1.Close();
        }
    }
}
```

When run, the example code prints the following to the system console:

```
Transaction 1 Transaction Isolation Level: ReadCommitted
Transaction 2 Transaction Isolation Level: Serializable
Transaction 3 Transaction Isolation Level: ReadCommitted
```

Reading Data (ADO.Net)

To read data from the database use `VerticaDataReader`, an implementation of `DbDataReader`. This implementation is useful for moving large volumes of data quickly off the server where it can be run through analytic applications.

Note: A `VerticaCommand` cannot execute anything else while it has an open `VerticaDataReader` associated with it. To execute something else, close the data reader or use a different `VerticaCommand` object.

To Read Data From the Database Using `VerticaDataReader`:

1. [Create a connection to the database.](#)
2. Create a command object using the connection.

```
VerticaCommand command = _conn.CreateCommand();
```

3. Create a query. This query works with the example VMart database.

```
command.CommandText =  
"SELECT fat_content, product_description " +  
"FROM (SELECT DISTINCT fat_content, product_description" +  
"      FROM product_dimension " +  
"      WHERE department_description " + "      IN ('Dairy') " +  
"      ORDER BY fat_content) AS food " +  
"LIMIT 10;";
```

4. Execute the reader to return the results from the query. The following command calls the `ExecuteReader` method of the `VerticaCommand` object to obtain the `VerticaDataReader` object.

```
VerticaDataReader dr = command.ExecuteReader();
```

5. Read the data. The data reader returns results in a sequential stream. Therefore, you must read data from tables row-by-row. The following example uses a while loop to accomplish this:

```
Console.WriteLine("\n\n Fat Content\t Product Description");  
Console.WriteLine("-----\t -----");
```

```
int rows = 0;
while (dr.Read())
{
    Console.WriteLine("      " + dr[0] + "      \t " + dr[1]);
    ++rows;
}
Console.WriteLine("-----\n (" + rows + " rows)\n");
```

6. When you're finished, close the data reader to free up resources.

```
dr.Close();
```

Example Usage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            VerticaCommand command = _conn.CreateCommand();
            command.CommandText =
                "SELECT fat_content, product_description " +
                "FROM (SELECT DISTINCT fat_content, product_description " +
                "      FROM product_dimension " +
                "      WHERE department_description " +
                "      IN ('Dairy') " +
                "      ORDER BY fat_content) AS food " +
                "LIMIT 10;";
            VerticaDataReader dr = command.ExecuteReader();

            Console.WriteLine("\n\n Fat Content\t Product Description");
            Console.WriteLine("-----\t -----");
            int rows = 0;
            while (dr.Read())
            {
                Console.WriteLine("      " + dr[0] + "      \t " + dr[1]);
                ++rows;
            }
            Console.WriteLine("-----\n (" + rows + " rows)\n");
            dr.Close();
            _conn.Close();
        }
    }
}
```

```
}  
  }  
}
```

Loading Data Through ADO.Net

This section details the different ways that you can load data in Vertica using the ADO.NET client driver:

- [Using the Vertica Data Adapter](#)
- [Example Batch Insert Using Parameters and Transactions](#)
- [Streaming Data Via ADO.NET](#)

Using the Vertica Data Adapter

The Vertica data adapter (VerticaDataAdapter) enables a client to exchange data between a data set and a Vertica database. It is an implementation of DbDataAdapter. You can use VerticaDataAdapter to simply read data, or, for example, read data from a database into a data set, and then write changed data from the data set back to the database.

Batching Updates

When using the `Update()` method to update a dataset, you can optionally use the `UpdateBatchSize()` method prior to calling `Update()` to reduce the number of times the client communicates with the server to perform the update. The default value of `UpdateBatchSize` is 1. If you have multiple `rows.Add()` commands for a data set, then you can change the batch size to an optimal size to speed up the operations your client must perform to complete the update.

Reading Data From Vertica Using the Data adapter:

The following example details how to perform a select query on the VMart schema and load the result into a DataTable, then output the contents of the DataTable to the console.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();

            // Try/Catch any exceptions
            try
            {
                using (_conn)
                {
                    // Create the command
                    VerticaCommand command = _conn.CreateCommand();
                    command.CommandText = "select product_key, product_description " +
                        "from product_dimension where product_key < 10";

                    // Associate the command with the connection
                    command.Connection = _conn;

                    // Create the DataAdapter
                    VerticaDataAdapter adapter = new VerticaDataAdapter();
                    adapter.SelectCommand = command;

                    // Fill the DataTable
                    DataTable table = new DataTable();
                    adapter.Fill(table);

                    // Display each row and column value.
                    int i = 1;
                    foreach (DataRow row in table.Rows)
                    {
                        foreach (DataColumn column in table.Columns)
                        {
                            Console.Write(row[column] + "\t");
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
        Console.WriteLine();
        i++;
    }
    Console.WriteLine(i + " rows returned.");
}
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
_conn.Close();
}
}
```

Reading Data From Vertica into a DataSet and Changing data:

The following example shows how to use a data adapter to read from and insert into a dimension table of the VMart schema.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using Vertica.Data.VerticaClient
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();

            // Try/Catch any exceptions
            try
            {
                using (_conn)
                {

                    //Create a data adapter object using the connection
                    VerticaDataAdapter da = new VerticaDataAdapter();

                    //Create a select statement that retrieves data from the table
                    da.SelectCommand = new
                    VerticaCommand("select * from product_dimension where product_key < 10",
                    _conn);
                    //Set up the insert command for the data adapter, and bind variables for
                    some of the columns
                    da.InsertCommand = new
                    VerticaCommand("insert into product_dimension values( :key, :version, :desc
                    )",
                    _conn);
                    da.InsertCommand.Parameters.Add(new VerticaParameter("key", VerticaType.BigInt));
                    da.InsertCommand.Parameters.Add(new VerticaParameter("version",
                    VerticaType.BigInt));
                    da.InsertCommand.Parameters.Add(new VerticaParameter("desc",
                    VerticaType.VarChar));
                    da.InsertCommand.Parameters[0].SourceColumn = "product_key";
                    da.InsertCommand.Parameters[1].SourceColumn = "product_version";
                    da.InsertCommand.Parameters[2].SourceColumn = "product_description";
                    da.TableMappings.Add("product_key", "product_key");
```

```
da.TableMappings.Add("product_version", "product_version");
da.TableMappings.Add("product_description", "product_description");

//Create and fill a Data set for this dimension table, and get the
resulting DataTable.
DataSet ds = new DataSet();
da.Fill(ds, 0, 0, "product_dimension");
DataTable dt = ds.Tables[0];

//Bind parameters and add two rows to the table.
DataRow dr = dt.NewRow();
dr["product_key"] = 838929;
dr["product_version"] = 5;
dr["product_description"] = "New item 5";
dt.Rows.Add(dr);
dr = dt.NewRow();
dr["product_key"] = 838929;
dr["product_version"] = 6;
dr["product_description"] = "New item 6";
dt.Rows.Add(dr);
//Extract the changes for the added rows.
DataSet ds2 = ds.GetChanges();

//Send the modifications to the server.
int updateCount = da.Update(ds2, "product_dimension");

//Merge the changes into the original Data set, and mark it up to date.
ds.Merge(ds2);
ds.AcceptChanges();
Console.WriteLine(updateCount + " updates made!");
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
_conn.Close();
}
}
```

Using Batch Inserts and Prepared Statements

You can load data in batches using a prepared statement with parameters. You can also use transactions to rollback the batch load if any errors are encountered.

If you are loading large batches of data (more than 100MB), then consider using a [direct batch insert](#).

The following example details using data contained in arrays, parameters, and a transaction to batch load data.

The test table used in the example is created with the command:

```
=> CREATE TABLE test (id INT, username VARCHAR(24), email VARCHAR(64), password VARCHAR(8));
```

Example Batch Insert Using Parameters and Transactions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            // Create arrays for column data
            int[] ids = {1, 2, 3, 4};
            string[] usernames = {"user1", "user2", "user3", "user4"};
            string[] emails = { "user1@example.com",
"user2@example.com","user3@example.com","user4@example.com" };
            string[] passwords = { "pass1", "pass2", "pass3", "pass4" };
            // create counters for accepted and rejected rows
            int rows = 0;
            int rejRows = 0;
            bool error = false;
            // Create the transaction
            VerticaTransaction txn = _conn.BeginTransaction();
            // Create the parameterized query and assign parameter types
            VerticaCommand command = _conn.CreateCommand();
            command.CommandText = "insert into TEST values (@id, @username, @email, @password)";
            command.Parameters.Add(new VerticaParameter("id", VerticaType.BigInt));
            command.Parameters.Add(new VerticaParameter("username", VerticaType.VarChar));
            command.Parameters.Add(new VerticaParameter("email", VerticaType.VarChar));
            command.Parameters.Add(new VerticaParameter("password", VerticaType.VarChar));
            // Prepare the statement
            command.Prepare();

            // Loop through the column arrays and insert the data
            for (int i = 0; i < ids.Length; i++) {
                command.Parameters["id"].Value = ids[i];
                command.Parameters["username"].Value = usernames[i];
                command.Parameters["email"].Value = emails[i];
                command.Parameters["password"].Value = passwords[i];
                try
                {
                    rows += command.ExecuteNonQuery();
                }
                catch (Exception e)
            }
        }
    }
}
```

```
        {
            Console.WriteLine("\nInsert failed - \n " + e.Message + "\n");
            ++rejRows;
            error = true;
        }
    }
    if (error)
    {
        // Roll back if errors
        Console.WriteLine("Errors. Rolling Back Transaction.");
        Console.WriteLine(rejRows + " rows rejected.");
        txn.Rollback();
    }
    else
    {
        // Commit if no errors
        Console.WriteLine("No Errors. Committing Transaction.");
        txn.Commit();
        Console.WriteLine("Inserted " + rows + " rows. ");
    }
    _conn.Close();
}
}
```

Loading Batches Directly into ROS

When loading large batches of data (more than 100MB or so), you should load the data directly into ROS containers. Inserting directly into ROS is more efficient for large loads than AUTO mode, since it avoids overflowing the WOS and spilling the remainder of the batch to ROS. Otherwise, the Tuple Mover has to perform a moveout on the data in the WOS, while subsequent data is directly written into ROS containers. This results in the data from your batch being segmented across containers.

When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see [Choosing a Load Method](#).

To directly load batches into ROS, set the `DirectBatchInsert` connection property to true. See [Opening and Closing the Database Connection](#) for details on all of the connection properties. When the `DirectBatchInsert` property is set to true, all batch inserts bypass the WOS and load directly into a ROS container.

Example usage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            builder.DirectBatchInsert = true;
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            //Perform some operations
            _conn.Close();
        }
    }
}
```

Streaming Data Via ADO.NET

There are two options to stream data from a file on the client to your Vertica database through ADO.NET:

- Use the `VerticaCopyStream` ADO.NET class to stream data in an object-oriented manner
- Execute a `COPY LOCAL` SQL statement to stream the data

The topics in this section explain how to use these options.

Streaming From the Client Via VerticaCopyStream

The `VerticaCopyStream` class lets you stream data from the client system to a Vertica database. It lets you use the SQL [COPY statement](#) directly without having to copy the data to a host in the database cluster first by substituting one or more data stream(s) for STDIN.

Notes:

- Use Transactions and disable auto commit on the copy command for better performance.
- Disable auto commit using the copy command with the 'no commit' modifier. You must explicitly disable commits. Enabling transactions does not disable autocommit when using `VerticaCopyStream`.
- The copy command used with `VerticaCopyStream` uses copy syntax.
- `VerticaCopyStream.rejects` is zeroed every time `execute` is called. If you want to capture the number of rejects, assign the value of `VerticaCopyStream.rejects` to another variable before calling `execute` again.
- You can add multiple streams using multiple `AddStream()` calls.

Example usage:

The following example demonstrates using `VerticaCopyStream` to copy a file stream into Vertica.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.IO;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Configure connection properties
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();

            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            //open the connection
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            try
            {
                using (_conn)
                {
                    // Start a transaction
                    VerticaTransaction txn = _conn.BeginTransaction();

                    // Create a table for this example
                    VerticaCommand command = new VerticaCommand("DROP TABLE IF EXISTS copy_
table", _conn);

                    command.ExecuteNonQuery();
                    command.CommandText = "CREATE TABLE copy_table (Last_Name char(50), "
                        + "First_Name char(50),Email char(50), "
                        + "Phone_Number char(15))";
                    command.ExecuteNonQuery();
                    // Create a new filestream from the data file
                    string filename = "C:/customers.txt";
                    Console.WriteLine("\n\nLoading File: " + filename);
                    FileStream inputfile = File.OpenRead(filename);
                    // Define the copy command
                    string copy = "copy copy_table from stdin record terminator E'\n'
delimiter '|' + " enforce length "
                        + " no commit";
                    // Create a new copy stream instance with the connection and copy statement
                    VerticaCopyStream vcs = new VerticaCopyStream(_conn, copy);

                    // Start the VerticaCopyStream process
                    vcs.Start();
                    // Add the file stream
```

```
        vcs.AddStream(inputfile, false);

        // Execute the copy
        vcs.Execute();

        // Finish stream and write out the list of inserted and rejected rows
        long rowsInserted = vcs.Finish();
        IList<long> rowsRejected = vcs.Rejects;
        // Does not work when rejected or exceptions defined
        Console.WriteLine("Number of Rows inserted: " + rowsInserted);
        Console.WriteLine("Number of Rows rejected: " + rowsRejected.Count);
        if (rowsRejected.Count > 0)
        {
            for (int i = 0; i < rowsRejected.Count; i++)
            {
                Console.WriteLine("Rejected row #{0} is row {1}", i, rowsRejected[i]);
            }
        }

        // Commit the changes
        txn.Commit();
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}

//close the connection
_conn.Close();
}
}
```

Using Copy with ADO.NET

To use COPY with ADO.NET, just execute a COPY statement and the path to the source file on the client system. This method is simpler than using the VerticaCopyStream class. However, you may prefer using VerticaCopyStream if you have many files to copy to the database or if your data comes from a source other than a local file (streamed over a network connection, for example).

The following example code demonstrates using COPY to copy a file from the client to the database. It is the same as the code shown in Bulk Loading Using the COPY Statement and the path to the data file is on the client system, rather than on the server.

To load data that is stored on a database node, use a VerticaCommand object to create a COPY command:

1. [Create a connection to the database](#) through the node on which the data file is stored.
2. Create a command object using the connection.

```
VerticaCommand command = _conn.CreateCommand();
```

3. Copy data. The following is an example of using the COPY command to load data. It uses the LOCAL modifier to copy a file local to the client issuing the command.

```
command.CommandText = "copy lcopy_table from '/home/dbadmin/customers.txt'"  
+ " record terminator E'\n' delimiter '|' "  
+ " enforce length ";  
  
Int32 insertedRows = command.ExecuteNonQuery();  
Console.WriteLine(insertedRows + " inserted.");
```

Example Usage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.IO;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Configure connection properties
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";

            // Open the connection
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            try
            {
                using (_conn)
                {

                    // Start a transaction
                    VerticaTransaction txn = _conn.BeginTransaction();

                    // Create a table for this example
                    VerticaCommand command = new VerticaCommand("DROP TABLE IF EXISTS lcopy_
table", _conn);
                    command.ExecuteNonQuery();
                    command.CommandText = "CREATE TABLE IF NOT EXISTS lcopy_table (Last_Name char
(50), "
                        + "First_Name char(50),Email char(50), "
                        + "Phone_Number char(15))";
                    command.ExecuteNonQuery();
                    // Define the copy command
                    command.CommandText = "copy lcopy_table from
'/home/dbadmin/customers.txt'"
                        + " record terminator E'\n' delimiter '|' "
                        + " enforcelength "
                        + " no commit";
                    // Execute the copy
                    Int32 insertedRows = command.ExecuteNonQuery();
                    Console.WriteLine(insertedRows + " inserted.");
                    // Commit the changes
                    txn.Commit();
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.Message);
            }
        }
    }
}
```

```
    }  
  
    // Close the connection  
    _conn.Close();  
  }  
}
```

Handling Messages (ADO.NET)

You can capture info and warning messages that Vertica provides to the ADO.NET driver by using the `InfoMessage` event on the `VerticaConnection` delegate class. This class captures messages that are not severe enough to force an exception to be triggered, but might still provide information that can benefit your application.

To Use the `VerticaInfoMessageEventHandler` class:

1. Create a method to handle the message sent from the even handler:

```
static void conn_InfoMessage(object sender, VerticaInfoMessageEventArgs e)  
{  
    Console.WriteLine(e.SqlState + ": " + e.Message);  
}
```

2. Create a [connection](#) and register a new `VerticaInfoMessageHandler` delegate for the `InfoMessage` event:

```
_conn.InfoMessage += new VerticaInfoMessageEventHandler(conn_InfoMessage);
```

3. Execute your queries. If a message is generated, then the event handle function is run.
4. You can unsubscribe from the event with the following command:

```
_conn.InfoMessage -= new VerticaInfoMessageEventHandler(conn_InfoMessage);
```

Example usage:

```
using System;  
using System.Collections.Generic;  
using System.Linq;
```

```
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
```

// define message handler to deal with messages

```
        static void conn_InfoMessage(object sender, VerticaInfoMessageEventArgs e)
        {
            Console.WriteLine(e.SqlState + ": " + e.Message);
        }
        static void Main(string[] args)
        {
```

//configure connection properties

```
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
```

//open the connection

```
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
```

//create message handler instance by subscribing it to the InfoMessage event of the connection

```
            _conn.InfoMessage += new VerticaInfoMessageEventHandler(conn_InfoMessage);
```

//create and execute the command

```
            VerticaCommand cmd = _conn.CreateCommand();
            cmd.CommandText = "drop table if exists fakeTable";
            cmd.ExecuteNonQuery();
```

//close the connection

```
            _conn.Close();
        }
    }
}
```

This examples displays the following when run:

```
00000: Nothing was dropped
```

Getting Table Metadata (ADO.Net)

You can get the table metadata by using the `GetSchema()` method on a connection and loading the metadata into a `DataTable`:

```
DataTable table = _conn.GetSchema("Tables", new string[] { database_name, schema_name, table_name, table_type });
```

For example:

```
DataTable table = _conn.GetSchema("Tables", new string[] { null, null, null, "SYSTEM TABLE" });
```

database_name, *schema_name*, *table_name* can be set to *null*, be a specific name, or use a LIKE pattern.

table_type can be one of:

- "SYSTEM TABLE"
- "TABLE"
- "GLOBAL TEMPORARY"
- "LOCAL TEMPORARY"
- "VIEW"
- null

If *table_type* is set to null, then the metadata for all metadata tables is returned.

Example Usage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
//configure connection properties
```

```
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
builder.Host = "192.168.1.10";
builder.Database = "VMart";
builder.User = "dbadmin";
```

//open the connection

```
VerticaConnection _conn = new VerticaConnection(builder.ToString());
_conn.Open();
```

//create a new data table containing the schema

```
//the last argument can be "SYSTEM TABLE", "TABLE", "GLOBAL TEMPORARY",
// "LOCAL TEMPORARY", "VIEW", or null for all types
DataTable table = _conn.GetSchema("Tables", new string[] { null, null, null, "SYSTEM
TABLE" });
```

//print out the schema

```
foreach (DataRow row in table.Rows)
{
    foreach (DataColumn col in table.Columns)
    {
        Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
    }
    Console.WriteLine("=====");
}
```

//close the connection

```
    _conn.Close();
}
}
```

ADO.NET Data Types

This table details the mapping between Vertica data type's and .NET and ADO.NET data types.

.NET Framework Type	ADO.NET DbType	VerticaType	Vertica Data Type	VerticaDataReader getter
Boolean	Boolean	Bit	Boolean	GetBoolean()
byte[]	Binary	Binary VarBinary LongVarBinar	Binary VarBinary LongVarBinar	GetBytes() Note: The limit for

.NET Framework Type	ADO.NET DbType	VerticaType	Vertica Data Type	VerticaDataReader getter
		y	y	<p>LongVarBinary is 32 Million bytes. If you attempt to insert more than the limit during a batch transfer for any one row, then they entire batch fails. Verify the size of the data before attempting to insert a LongVarBinary during a batch.</p>
Datetime	DateTime	Date Time TimeStamp	Date Time TimeStamp	<p>GetDateTime()</p> <p>Note: The Time portion of the DateTime object for vertica dates is set to DateTime.MinValue. Previously, VerticaType.DateTime was used for all date/time types. VerticaType.DateTime still exists for backwards compatibility, but now there are more specific VerticaTypes for each type.</p>
DateTimeOffset	DateTimeOffset	TimestampTZ	TimestampTZ	GetDateTimeOffset()

.NET Framework Type	ADO.NET DbType	VerticaType	Vertica Data Type	VerticaDataReader getter
		TimeTZ	TimeTZ	Note: The Date portion of the DateTime is set to DateTime.MinValue
Decimal	Decimal	Numeric	Numeric	GetDecimal()
Double	Double	Double	Double Precision	GetDouble() Note: Vertica Double type uses a default precision of 53.
Int64	Int64	BigInt	Integer	GetInt64()
TimeSpan	Object	13 Interval Types	13 Interval Types	GetInterval() Note: There are 13 VerticaType values for the 13 types of intervals. The specific VerticaType used determines the conversion rules that the driver applies. Year/Month intervals represented as 365/30 days
String	String	Varchar LongVarChar	Varchar LongVarChar	GetString()
String	StringFixedLength	Char	Char	GetString()

.NET Framework Type	ADO.NET DbType	VerticaType	Vertica Data Type	VerticaDataReader getter
Object	Object	N/A	N/A	GetValue()

Programming Python Client Applications

In order to use Python with Vertica, you must either install the Vertica Python Client (see [Python Client Documentation](#)), or install the pyodbc module and a Vertica ODBC driver on the machine where Python is installed. See [Python Prerequisites](#).

Python on Linux

Most Linux distributions come with Python preinstalled. If you want a more recent version, you can download and build it from the source code, though sometimes RPMs are also available. See the [Python Web site](#) and click an individual release for details. See also [Python documentation](#).

To determine the Python version on your Linux operating systems, type the following at a command prompt:

```
# python -V
```

The system returns the version; for example:

```
Python 3.3.4
```

Python on Windows

Python is not required to run natively on Windows operating systems, so it is not preinstalled. The ActiveState Web site distributes a free Windows installer for Python called [ActivePython](#).

If you need installation instructions for Windows, see [Using Python on Windows](#) at python.org. [Python on Windows](#) at diveintopython.org provides installation instructions for both the ActivePython and python.org packages.

Python and Unicode

When you are using Python, be sure that all of your components are using the same unicode text encoding. By default, the DSN Parameter [ColumnsAsChar](#) causes the ODBC driver to report CHAR and VARCHAR values as SQL_WCHAR. The driver returns these values to the driver manager in the encoding expected by the driver manager, as controlled by the

DriverManagerEncoding parameter in vertica.ini. Similarly, your Python application must use the encoding expected by the driver manager. If any of these components use different encodings, your output can become garbled.

The Vertica Python Client

Vertica has a native Python client you can use to communicate with your Vertica database.

Before you can connect to Vertica using Python, you need to download the Python Client. See the Vertica Python Client Documentation, for download and installation instructions.

Using pyodbc and Vertica

Before you can connect to Vertica using pyodbc, you need to download the pyodbc module, which communicates with iODBC/unixODBC driver on UNIX operating systems and the ODBC Driver Manager for Windows operating systems.

The pyodbc module is an open source , MIT-licensed Python module, letting you use ODBC to connect to almost any database from Windows, Linux, Mac OS/X, and other operating systems.

Vertica supports multiple versions of pyodbc. See [Python Prerequisites](#) for additional details.

Download the source distribution from the [pyodbc Web site](#), unpack it and build it. Note that you need the unixODBC development package (in addition to the regular build tools) to build pyodbc. For example, on RedHat/CentOS run: `yum install unixODBC-devel`, and on Ubuntu run: `sudo apt-get install unixodbc-dev`. See the [pyodbc wiki](#) for detailed instructions.

External Resources

- [Python Database API Specification v2.0](#)
- [Python documentation](#)

Configuring the ODBC Run-Time Environment on Linux

To configure the ODBC run-time environment on Linux:

1. Create the `odbc.ini` file if it does not already exist.
2. Add the ODBC driver directory to the `LD_LIBRARY_PATH` system environment variable:

```
export LD_LIBRARY_PATH=/path-to-vertica-odbc-driver:$LD_LIBRARY_PATH
```

Important: If you skip Step 2, the ODBC manager cannot find the driver in order to load it.

These steps are relevant only for unixODBC and iODBC. See their respective documentation for details on `odbc.ini`.

See Also

- [unixODBC Web site](#)
- [iODBC Web site](#)

Querying the Database Using Python and pyodbc

The example session below uses `pyodbc` with the Vertica ODBC driver to connect Python to the Vertica database.

Note: `SQLFetchScroll` and `SQLFetch` functions cannot be mixed together in iODBC code. When using `pyodbc` with the iODBC driver manager, `skip` cannot be used with the `fetchall`, `fetchone`, and `fetchmany` functions.

Example Script

The following example script shows how to query Vertica using Python 3, `pyodbc`, and an ODBC DSN.

```
import pyodbc
cnxn = pyodbc.connect("DSN=VerticaDSN", ansi=True)
cursor = cnxn.cursor()
# create table
cursor.execute("CREATE TABLE TEST("
    "C_ID INT,"
    "C_FP FLOAT,"
    "C_VARCHAR VARCHAR(100),"
    "C_DATE DATE, C_TIME TIME,")
```

```
"C_TS TIMESTAMP,"
"C_BOOL BOOL)"
cursor.execute("INSERT INTO test VALUES(1,1.1,'abcdefg1234567890','1901-01-01','23:12:34','1901-01-01
09:00:09','t')")
cursor.execute("INSERT INTO test VALUES(2,3.4,'zxcasdqwe09876543','1991-11-11','00:00:01','1981-12-31
19:19:19','f')")
cursor.execute("SELECT * FROM TEST")
rows = cursor.fetchall()
for row in rows:
    print(row, end='\n')
cursor.execute("DROP TABLE TEST CASCADE")
cursor.close()
cnxn.close()
```

The resulting output displays:

```
(2, 3.4, 'zxcasdqwe09876543', datetime.date(1991, 11, 11), datetime.time(0, 0, 1), datetime.datetime
(1981, 12, 31, 19, 19, 19), False)
(1, 1.1, 'abcdefg1234567890', datetime.date(1901, 1, 1), datetime.time(23, 12, 34), datetime.datetime
(1901, 1, 1, 9, 0, 9), True)
```

Notes

`SQLPrimaryKeys` returns the table name in the primary (`pk_name`) column for unnamed primary constraints. For example:

- Unnamed primary key:

```
CREATE TABLE schema.test(c INT PRIMARY KEY);

SQLPrimaryKeys
"TABLE_CAT", "TABLE_SCHEM", "TABLE_NAME", "COLUMN_NAME", "KEY_SEQ", "PK_NAME" <Null>, "SCHEMA",
"TEST", "C", 1, "TEST"
```

- Named primary key:

```
CREATE TABLE schema.test(c INT CONSTRAINT pk_1 PRIMARY KEY);

SQLPrimaryKeys
"TABLE_CAT", "TABLE_SCHEM", "TABLE_NAME", "COLUMN_NAME", "KEY_SEQ", "PK_NAME" <Null>, "SCHEMA",
"TEST", "C", 1, "PK_1"
```

Micro Focus recommends that you name your constraints.

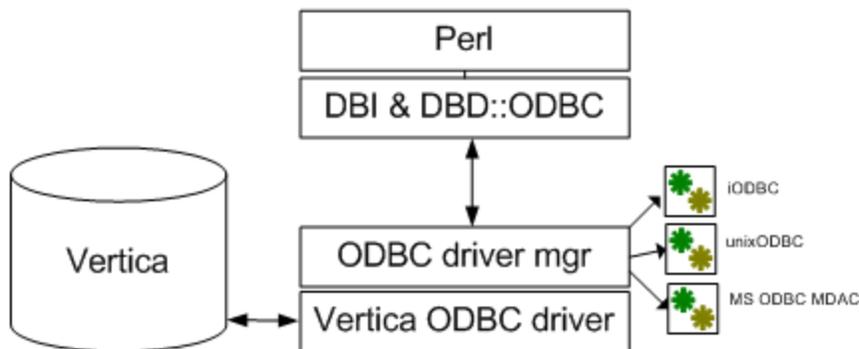
See Also

- [Loading Data Through ODBC](#)

Programming Perl Client Applications

The Perl programming language has a Database Interface module (DBI) that creates a standard interface for Perl scripts to interact with databases. The interface module relies on Database Driver modules (DBDs) to handle all of the database-specific communication tasks. The result is an interface that provides a consistent way for Perl scripts to interact with many different types of databases.

Your Perl script can interact with Vertica using the Perl DBI module along with the DBD::ODBC database driver to interface to Vertica's ODBC driver. See the CPAN pages for Perl's [DBI](#) and [DBD::ODBC](#) modules for detailed documentation.



Important: With Perl ODBC clients, Vertica allows a forked process (a child process) to drop the parent connection to the Vertica server when the child process completes and exits. Vertica allows this behavior regardless of the setting of the Perl DBI `AutoInactiveDestroy` attribute. To change the default setting so that Vertica honors the setting of the Perl DBI `AutoInactiveDestroy` attribute, add the parameter `CleanupInForkChild` to your `vertica.ini` file, and set its value to 1. When the Perl DBI `AutoInactiveDestroy` attribute is set to 1, and the Vertica parameter `CleanupInForkChild` is set to 1, Vertica does not drop the parent connection upon child process completion.

The topics in this chapter explain how to:

- Configure Perl to access Vertica
- Connect to Vertica
- Query data stored in Vertica
- Insert data into Vertica

Perl Client Prerequisites

In order run a Perl client script that connects to Vertica, your client system must have:

- The Vertica ODBC drivers installed and configured. See [Installing the Vertica Client Drivers](#) for details.
- A Data Source Name (DSN) containing the connection parameters for your Vertica. See [Creating an ODBC Data Source Name](#). (Optionally, your Perl script can connect to Vertica without using a DSN as described in [Connecting From Perl Without a DSN](#)).
- A supported version of Perl installed
- The DBI and DBD::ODBC Perl modules (see below)

Supported Perl Versions

Vertica supports Perl versions 5.8 and 5.10. Versions later than 5.10 may also work.

Perl on Linux

Most Linux distributions come with Perl preinstalled. See your Linux distribution's documentation for details of installing and configuring its Perl package if it is not already installed.

To determine the Perl version on your Linux operating systems, type the following at a command prompt:

```
# perl -v
```

The system returns the version; for example:

```
This is perl, v5.10.0 built for x86_64-linux-thread-multi
```

Perl on Windows

Perl is not installed by default on Windows platforms. There are several different Perl packages you can download and install on your Windows system:

- [ActivePerl](#) by Activestate is a commercially-supported version of Perl for Windows platforms.
- [Strawberry Perl](#) is an open-source port of Perl for Windows.

The Perl Driver Modules (DBI and DBD::ODBC)

Before you can connect to Vertica using Perl, your Perl installation needs to have the Perl Database Interface module (DBI) and the Database Driver for ODBC (DBD::ODBC). These modules communicate with iODBC/unixODBC driver on UNIX operating systems or the ODBC Driver Manager for Windows operating systems.

Vertica supports the following Perl modules:

- DBI version 1.609 (DBI-1.609.tar.gz)
- DBD::ODBC version 1.22 (DBD-ODBC-1.22.tar.gz)

Later versions of DBI and DBD::ODBC may also work.

DBI is installed by default with many Perl installations. You can test whether it is installed by executing the following command on the Linux or Windows command line:

```
# perl -e "use DBI;"
```

If the command exits without printing anything, then DBI is installed. If it prints an error, such as:

```
Can't locate DBI.pm in @INC (@INC contains: /usr/local/lib64/perl5/usr/local/share/perl5
/usr/lib64/perl5/vendor_perl /usr/share/perl5/vendor_perl
/usr/lib64/perl5 /usr/share/perl5 .) at -e line 1.
BEGIN failed--compilation aborted at -e line 1.
```

then DBI is not installed.

Similarly, you can see if DBD::ODBC is installed by executing the command:

```
# perl -e "use DBD::ODBC;"
```

You can also run the following Perl script to determine if DBI and DBD::ODBC are installed. If they are, the script lists any available DSNs.

```
#!/usr/bin/perl
use strict;
# Attempt to load the DBI module in an eval using require. Prevents
# script from erroring out if DBI is not installed.
eval
{
    require DBI;
    DBI->import();
};
if ($?) {
    # The eval failed, so DBI must not be installed
    print "DBI module is not installed\n";
} else {
    # Eval was successful, so DBI is installed
    print "DBI Module is installed\n";
    # List the drivers that DBI knows about.
    my @drivers = DBI->available_drivers;
    print "Available Drivers: \n";
    foreach my $driver (@drivers) {
        print "\t$driver\n";
    }
    # See if DBD::ODBC is installed by searching driver array.
    if (grep {/ODBC/i} @drivers) {
        print "\nDBD::ODBC is installed.\n";
        # List the ODBC data sources (DSNs) defined on the system
        print "Defined ODBC Data Sources:\n";
        my @dsns = DBI->data_sources('ODBC');
        foreach my $dsn (@dsns) {
            print "\t$dsn\n";
        }
    } else {
        print "DBD::ODBC is not installed\n";
    }
}
}
```

The exact output of the above code will depend on the configuration of your system. The following is an example of running the code on a Windows computer:

```
DBI Module is installed
Available Drivers:
    ADO
    DBM
    ExampleP
    File
    Gofer
    ODBC
    Pg
    Proxy
    SQLite
    Sponge
    mysql
DBD::ODBC is installed.
Defined ODBC Data Sources:
    dbi:ODBC:dBASE Files
    dbi:ODBC:Excel Files
```

```
dbi:ODBC:MS Access Database  
dbi:ODBC:VerticaDSN
```

Installing Missing Perl Modules

If Perl's DBI or DBD::ODBC modules are not installed on your client system, you must install them before your Perl scripts can connect to Vertica. How you install modules depends on your Perl configuration:

- For most Perl installations, you use the `cpan` command to install modules. If the `cpan` command alias isn't installed on your system, you can try to start CPAN by using the command:

```
perl -MCPAN -e shell
```

- Some Linux distributions provide Perl modules as packages that can be installed with the system package manager (such as `yum` or `apt`). See your Linux distribution's documentation for details.
- On ActiveState Perl for Windows, you use the Perl Package Manager (PPM) program to install Perl modules. See the [Activestate's PPM documentation](#) for details.

Note: Installing Perl modules usually requires administrator or root privileges. If you do not have these permissions on your client system, you need to ask your system administrator to install these modules for you.

Connecting to Vertica Using Perl

You use the Perl DBI module's `connect` function to connect to Vertica. This function takes a required data source string argument and optional arguments for the username, password, and connection attributes.

The data source string must start with `"dbi:ODBC:"`, which tells the DBI module to use the `DBD::ODBC` driver to connect to Vertica. The remainder of the string is interpreted by the `DBD::ODBC` driver. It usually contains the name of a DSN that contains the connection information needed to connect to your Vertica database. For example, to tell the `DBD::ODBC` driver to use the DSN named `VerticaDSN`, you use the data source string:

```
"dbi:ODBC:VerticaDSN"
```

The username and password parameters are optional. However, if you do not supply them (or just the username for a passwordless account) and they are not set in the DSN, attempting to connect always fails.

The `connect` function returns a database handle if it connects to Vertica. If it does not, it returns `undef`. In that case, you can access the DBI module's error string property (`$DBI::errstr`) to get the error message.

Note: By default, the DBI module prints an error message to `STDERR` whenever it encounters an error. If you prefer to display your own error messages or handle errors in some other manner, you may want to disable these automatic messages by setting DBI's `PrintError` connection attribute to `false`. See [Setting Perl DBI Connection Attributes](#) for details. Otherwise, users may see two error messages: the one that DBI prints automatically, and the one that your script prints on its own.

The following example demonstrates connecting to Vertica using a DSN named `VerticaDSN`. The call to `connect` supplies a username and password. After connecting, it calls the database handle's `disconnect` function, which closes the connection.

```
#!/usr/bin/perl -w
use strict;
use DBI;
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123");
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
print "Connected!\n";
$dbh->disconnect();
```

Setting ODBC Connection Parameters in Perl

To set ODBC connection parameters, replace the DSN name with a semicolon delimited list of parameter name and value pairs in the source data string. Use the DSN parameter to tell `DBD::ODBC` which DSN to use, then add in other the other ODBC parameters you want to set. For example, the following code connects using a DSN named `VerticaDSN` and sets the connection's locale to `en_GB`.

```
#!/usr/bin/perl -w
use strict;
use DBI;
# Instead of just using the DSN name, use name and value pairs.
my $dbh = DBI->connect("dbi:ODBC:DSN=VerticaDSN;Locale=en_
GB@collation=binary","ExampleUser","password123");
unless (defined $dbh) {
    # Connection failed.
```

```
    die "Failed to connect: $DBI::errstr";  
}  
print "Connected!\n";  
$dbh->disconnect();
```

See [Data Source Name \(DSN\) Connection Properties](#) for a list of the connection parameters you can set in the source data string.

Setting Perl DBI Connection Attributes

The Perl DBI module has attributes that you can use to control the behavior of its database connection. These attributes are similar to the ODBC connection parameters (in several cases, they duplicate each other's functionality). The DBI connection attributes are a cross-platform way of controlling the behavior of the database connection.

You can set the DBI connection attributes when establishing a connection by passing the DBI connect function a hash containing attribute and value pairs. For example, to set the DBI connection attribute `AutoCommit` to false, you would use:

```
# Create a hash that holds attributes for the connection  
my $attr = {AutoCommit => 0};  
# Open a connection using a DSN. Supply the username and password.  
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",  
    $attr);
```

See the DBI documentation's [Database Handle Attributes](#) section for a full description of the attributes you can set on the database connection.

After your script has connected, it can access and modify the connection attributes through the database handle by using it as a hash reference. For example:

```
print "The AutoCommit attribute is: " . $dbh->{AutoCommit} . "\n";
```

The following example demonstrates setting two connection attributes:

- `RaiseError` controls whether the DBI driver generates a Perl error if it encounters a database error. Usually, you set this to true (1) if you want your Perl script to exit if there is a database error.
- `AutoCommit` controls whether statements automatically commit their transactions when they complete. DBI defaults to Vertica's default `AutoCommit` value of true. Always set `AutoCommit` to false (0) when bulk loading data to increase database efficiency.

```
#!/usr/bin/perl  
use strict;  
use DBI;
```

```
# Create a hash that holds attributes for the connection
my $attr = {
    RaiseError => 1, # Make database errors fatal to script
    AutoCommit => 0, # Prevent statements from committing
                    # their transactions.
};
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
    $attr);

if (defined $dbh->err) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
print "Connected!\n";
# The database handle lets you access the connection attributes directly:
print "The AutoCommit attribute is: " . $dbh->{AutoCommit} . "\n";
print "The RaiseError attribute is: " . $dbh->{RaiseError} . "\n";
# And you can change values, too...
$dbh->{AutoCommit} = 1;
print "The AutoCommit attribute is now: " . $dbh->{AutoCommit} . "\n";
$dbh->disconnect();
```

The example outputs the following when run:

```
Connected!The AutoCommit attribute is: 0
The RaiseError attribute is: 1
The AutoCommit attribute is now: 1
```

Connecting From Perl Without a DSN

If you do not want to set up a Data Source Name (DSN) for your database, you can supply all of the information Perl's DBD::ODBC driver requires to connect to your Vertica database in the data source string. This source string must the DRIVER= parameter that tells DBD::ODBC which driver library to use in order to connect. The value for this parameter is the name assigned to the driver by the client system's driver manager:

- On Windows, the name assigned to the Vertica ODBC driver by the driver manager is Vertica.
- On Linux and other UNIX-like operating systems, the Vertica ODBC driver's name is assigned in the system's `odbcinst.ini` file. For example, if your `/etc/odbcint.ini` contains the following:

```
[Vertica]
Description = Vertica ODBC Driver
Driver = /opt/vertica/lib64/libverticaodbc.so
```

you would use the name Vertica. See [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#) for more information about the `odbcinst.ini` file.

You can take advantage of Perl's variable expansion within strings to use variables for most of the connection properties as the following example demonstrates.

```
#!/usr/bin/perl
use strict;
use DBI;
my $server='VerticaHost';
my $port = '5433';
my $database = 'VMart';
my $user = 'ExampleUser';
my $password = 'password123';
# Connect without a DSN by supplying all of the information for the connection.
# The DRIVER value on UNIX platforms depends on the entry in the odbcinst.ini
# file.
my $dbh = DBI->connect("dbi:ODBC:DRIVER={Vertica};Server=$server;" .
    "Port=$port;Database=$database;UID=$user;PWD=$password")
    or die "Could not connect to database: " . DBI::errstr;
print "Connected!\n";
$dbh->disconnect();
```

Note: Surrounding the driver name with braces (`{` and `}`) in the source string is optional.

Executing Statements Using Perl

Once your Perl script has connected to Vertica (see [Connecting to Vertica Using Perl](#)), it can execute simple statements that return a value rather than a result set by using the Perl DBI module's `do` function. You usually use this function to execute DDL statements or data loading statements such as `COPY` (see [Using COPY LOCAL to Load Data in Perl](#)).

```
#!/usr/bin/perl
use strict;
use DBI;
# Disable autocommit
my $attr = {AutoCommit => 0};
# Open a connection using a DSN.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
    $attr);
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
# You can use the do function to perform DDL commands.
# Drop any existing table.
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;");
# Create a table to hold data.
$dbh->do("CREATE TABLE TEST( \
    C_ID INT, \
```

```
C_FP FLOAT,\
C_VARCHAR VARCHAR(100),\
C_DATE DATE, C_TIME TIME,\
C_TS TIMESTAMP,\
C_BOOL BOOL)");
# Commit changes and exit.
$dbh->commit();
$dbh->disconnect();
```

Note: The `do` function returns the number of rows that were affected by the statement (or -1 if the count of rows doesn't apply or is unavailable). Usually, the only time you need to consult this value is after you deleted a number of rows or if you used a bulk load command such as [COPY](#). You use other DBI functions instead of `do` to perform batch inserts and selects (see [Batch Loading Data Using Perl](#) and [Querying Using Perl](#) for details).

Batch Loading Data Using Perl

To load large batches of data into Vertica using Perl:

1. Set DBI's `AutoCommit` connection attribute to false to improve the batch load speed. See [Setting Perl DBI Connection Attributes](#) for an example of disabling `AutoCommit`.
2. Call the database handle's `prepare` function to prepare a SQL [INSERT](#) statement that contains placeholders for the data values you want to insert. For example:

```
# Prepare an INSERT statement for the test table
$stmt = $dbh->prepare("INSERT into test values(?,?,?,?);");
```

The `prepare` function returns a statement handle that you will use to insert the data.

3. Assign data to the placeholders. There are several ways to do this. The easiest is to populate an array with a value for each placeholder in your `INSERT` statement.
4. Call the statement handle's `execute` function to insert a row of data into Vertica. The return value of this function call lets you know whether Vertica accepted or rejected the row.
5. Repeat steps 3 and 4 until you have loaded all of the data you need to load.
6. Call the database handle's `commit` function to commit the data you inserted.

The following example demonstrates inserting a small batch of data by populating an array of arrays with data, then looping through it and inserting each row.

```
#!/usr/bin/perl
use strict;
use DBI;
# Create a hash reference that holds a hash of parameters for the
# connection.
my $attr = {AutoCommit => 0, # Turn off autocommit
            PrintError => 0 # Turn off automatic error printing.
            # This is handled manually.
            };
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
    $attr);
if (defined DBI::err) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
print "Connection AutoCommit state is: " . $dbh->{AutoCommit} . "\n";
# Create table to hold inserted data
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;") or die "Could not drop table";
$dbh->do("CREATE TABLE TEST( \
    C_ID INT, \
    C_FP FLOAT,\
    C_VARCHAR VARCHAR(100),\
    C_DATE DATE, C_TIME TIME,\
    C_TS TIMESTAMP,\
    C_BOOL BOOL)") or die "Could not create table";
# Populate an array of arrays with values. One of these rows contains
# data that will not be successfully inserted. Another contains an
# undef value, which gets inserted into the database as a NULL.
my @data = (
    [1,1.111,'Hello World!','2001-01-01','01:01:01'
     , '2001-01-01 01:01:01','t'],
    [2,2.22222,'How are you?','2002-02-02','02:02:02'
     , '2002-02-02 02:02:02','f'],
    ['bad value',2.22222,'How are you?','2002-02-02','02:02:02'
     , '2002-02-02 02:02:02','f'],
    [4,4.22222,undef,'2002-02-02','02:02:02'
     , '2002-02-02 02:02:02','f'],
    );
# Create a prepared statement to use parameters for inserting values.
my $sth = $dbh->prepare_cached("INSERT into test values(?,?,?,?,?)");
my $rowcount = 0; # Count # of rows
# Loop through the arrays to insert values
foreach my $tuple (@data) {
    $rowcount++;
    # Insert the row
    my $retval = $sth->execute(@$tuple);

    # See if the row was successfully inserted.
    if ($retval == 1) {
        # Value of 1 means the row was inserted (1 row was affected by insert)
        print "Row $rowcount successfully inserted\n";
    } else {
        print "Inserting row $rowcount failed";
        # Error message is not set on some platforms/versions of DBUI. Check to
        # ensure a message exists to avoid getting an uninitialized var warning.
        if ($sth->err()) {
            print ": " . $sth->errstr();
        }
        print "\n";
    }
}
```

```
    }  
  }  
  # Commit changes. With AutoCommit off, you need to use commit for batched  
  # data to actually be committed into the database. If your Perl script exits  
  # without committing its data, Vertica rolls back the transaction and the  
  # data is not committed.  
  $dbh->commit();  
  $dbh->disconnect();
```

The previous example displays the following when successfully run:

```
Connection AutoCommit state is: 0  
Row 1 successfully inserted  
Row 2 successfully inserted  
Inserting row 3 failed with error 01000 [Vertica][VerticaDSII] (20) An  
error occurred during query execution: Row rejected by server; see  
server log for details (SQL-01000)  
Row 4 successfully inserted
```

Note that one of the rows was not inserted because it contained a string value that could not be stored in an integer column. See [Conversions Between Perl and Vertica Data Types](#) for details of data type handling in Perl scripts that communicate with Vertica.

Using COPY LOCAL to Load Data in Perl

If you have delimited files (for example, a file with comma-separated values) on your client system that you want to load into Vertica, you can use the [COPY LOCAL](#) statement to directly load the file's contents into Vertica instead of using Perl to read, parse, and then batch insert the data. You execute a COPY LOCAL statement to load the file from the local filesystem. The result of executing the statement is the number of rows that were successfully inserted.

The following example code demonstrates loading a file named data.txt and located in the same directory as the Perl file into Vertica using a COPY LOCAL statement.

```
#!/usr/bin/perl  
use strict;  
use DBI;  
# Filesystem path handling module  
use File::Spec;  
# Create a hash reference that holds a hash of parameters for the  
# connection.  
my $attr = {AutoCommit => 0}; # Turn off AutoCommit  
# Open a connection using a DSN. Supply the username and password.  
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",  
    $attr) or die "Failed to connect: $DBI::errstr";  
print "Connected!\n";  
# Drop any existing table.  
$dbh->do("DROP TABLE IF EXISTS Customers CASCADE;");  
# Create a table to hold data.  
$dbh->do("CREATE TABLE Customers( \
```



```
7|Haviva|Hopper|Morgan@porttitor.edu|1975-05-10
8|Stewart|Sweeney|Rhonda@lectus.us|2003-06-20
9|Allen|Rogers|Alexander@enim.gov|2006-06-17
10|Trevor|Dillon|Eagan@id.org|1988-11-27
11|Leroy|Ashley|Carter@turpis.edu|1958-07-25
12|Elmo|Malone|Carla@enim.edu|1978-08-29
13|Laurel|Ball|Zelenia@Integer.us|1989-09-20
14|Zeus|Phillips|Branden@blandit.gov|1996-08-08
15|Alexis|McClean|Flavia@Suspendisse.org|2008-01-07
```

The example code produces the following output when run on a large sample file:

```
Connected!
Loading file /home/dbadmin/Perl/data.txt
Copied 1000000 rows into database.
ID  First      Last      EMail      Birthday
==  =====  =====  =====  ===========
 1  Georgia    Gomez     Rhiannon@magna.us    1937-10-03
 2  Abdul      Alexander Kathleen@ipsum.gov     1941-03-10
 3  Nigel      Contreras Tanner@et.com          1955-06-01
 4  Gray       Holt      Thomas@Integer.us     1945-12-06
 5  Candace    Bullock   Scott@vitae.gov        1932-05-27
 6  Matthew    Dotson    Keith@Cras.com         1956-09-30
 7  Haviva     Hopper    Morgan@porttitor.edu   1975-05-10
 8  Stewart    Sweeney   Rhonda@lectus.us      2003-06-20
 9  Allen      Rogers    Alexander@enim.gov     2006-06-17
10  Trevor     Dillon    Eagan@id.org           1988-11-27
11  Leroy      Ashley    Carter@turpis.edu      1958-07-25
12  Elmo       Malone    Carla@enim.edu         1978-08-29
13  Laurel     Ball      Zelenia@Integer.us     1989-09-20
14  Zeus       Phillips  Branden@blandit.gov    1996-08-08
```

Note: Loading a single, large data file into Vertica through a single data connection is less efficient than loading a number of smaller files onto multiple nodes in parallel. Loading onto multiple nodes prevents any one node from becoming a bottleneck.

Querying Using Perl

To query Vertica using Perl:

1. Prepare a query statement using the Perl DBI module's `prepare` function. This function returns a statement handle that you use to execute the query and get the result set.
2. Execute the prepared statement by calling the `execute` function on the statement handle.
3. Retrieve the results of the query from the statement handle using one of several methods, such as calling the statement handle's `fetchrow_array` function to retrieve a row of data, or `fetchall_array` to get an array of arrays containing the entire result set (not a good idea if your result set may be very large!).

The following example demonstrates querying the table created by the example shown in [Batch Loading Data Using Perl](#). It executes a query to retrieve all of the content of the table, then repeatedly calls the statement handle's `fetchrow_array` function to get rows of data in an array. It repeats this process until `fetchrow_array` returns `undef`, which means that there are no more rows to be read.

```
#!/usr/bin/perl
use strict;
use DBI;
my $attr = {RaiseError => 1 }; # Make errors fatal to the Perl script.
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
                      $attr);
# Prepare a query to get the content of the table
my $sth = $dbh->prepare("SELECT * FROM TEST ORDER BY C_ID ASC");
# Execute the query by calling execute on the statement handle
$sth->execute();
# Loop through result rows while we have them, getting each row as an array
while (my @row = $sth->fetchrow_array()) {
    # The @row array contains the column values for this row of data
    # Loop through the column values
    foreach my $column (@row) {
        if (!defined $column) {
            # NULLs are signaled by undefs. Set to NULL for clarity
            $column = "NULL";
        }
        print "$column\t"; # Output the column separated by a tab
    }
    print "\n";
}
$dbh->disconnect();
```

The example prints the following when run:

1	1.111	Hello World!	2001-01-01	01:01:01	2001-01-01	01:01:01	1
2	2.22222	How are you?	2002-02-02	02:02:02	2002-02-02	02:02:02	0
4	4.22222	NULL	2002-02-02	02:02:02	2002-02-02	02:02:02	0

Binding Variables to Column Values

Another method of retrieving the query results is to bind variables to columns in the result set using the statement handle's `bind_columns` function. You may find this method convenient if you need to perform extensive processing on the returned data, since your code can use variables rather than array references to access the data. The following example demonstrates binding variables to the result set, rather than looping through the row and column values.

```
#!/usr/bin/perl
use strict;
use DBI;
my $attr = {RaiseError => 1 }; # Make SQL errors fatal to the Perl script.
# Open a connection using a DSN. Supply the username and password.
```

```
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN32","ExampleUser","password123",
                      $attr);
# Prepare a query to get the content of the table
my $sth = $dbh->prepare("SELECT * FROM TEST ORDER BY C_ID ASC");
$sth->execute();
# Create a set of variables to bind to the column values.
my ($C_ID, $C_FP, $C_VARCHAR, $C_DATE, $C_TIME, $C_TS, $C_BOOL);
# Bind the variable references to the columns in the result set.
$sth->bind_columns(\$C_ID, \$C_FP, \$C_VARCHAR, \$C_DATE, \$C_TIME,
                  \$C_TS, \$C_BOOL);

# Now, calling fetch() to get a row of data updates the values of the bound
# variables. Continue calling fetch until it returns undefined.
while ($sth->fetch()) {
    # Note, you should always check that values are defined before using them,
    # since NULL values are translated into Perl as undefined. For this
    # example, just check the VARCHAR column for undefined values.
    if (!defined $C_VARCHAR) {
        $C_VARCHAR = "NULL";
    }
    # Just print values separated by tabs.
    print "$C_ID\t$C_FP\t$C_VARCHAR\t$C_DATE\t$C_TIME\t$C_TS\t$C_BOOL\n";
}
$dbh->disconnect();
```

The output of this example is identical to the output of the previous example.

Preparing, Querying, and Returning a Single Row

If you expect a single row as the result of a query (for example, when you execute a [COUNT \(*\)](#) query), you can use the DBI module's `selectrow_array` function to combine executing a statement and retrieving an array as a result.

The following example shows using `selectrow_array` to execute and get the results of the [SHOW LOCALE](#) statement. It also demonstrates changing the locale using the `do` function.

```
#!/usr/bin/perl
use strict;
use DBI;
my $attr = {RaiseError => 1 }; # Make SQL errors fatal to the Perl script.
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
                      $attr);
# Demonstrate setting/getting locale.
# Use selectrow_array to combine preparing a statement, executing it, and
# getting an array as a result.
my @localerv = $dbh->selectrow_array("SHOW LOCALE;");
# The locale name is the 2nd column (array index 1) in the result set.
print "Locale: $localerv[1]\n";
# Use do() to execute a SQL statement to set the locale.
$dbh->do("SET LOCALE TO en_GB");
# Get the locale again.
@localerv = $dbh->selectrow_array("SHOW LOCALE;");
print "Locale is now: $localerv[1]\n";
```

```
$dbh->disconnect();
```

The result of running the example is:

```
Locale: en_US@collation=binary (LEN_KBINARY)  
Locale is now: en_GB (LEN)
```

Executing Queries and ResultBufferSize Settings

When you call the `execute()` function on a prepared statement, the client library retrieves results up to the size of the result buffer. The result buffer size is set using ODBC's [ResultBufferSize](#) setting.

Vertica does not allow multiple active queries per connection. However, you can simulate multiple active queries by setting the result buffer to be large enough to accommodate the entire results from the first query. To ensure that the ODBC client driver's buffer is large enough to store result set for first query you can set `ResultBufferSize` to 0. Setting this parameter to 0 makes the result buffer size unlimited. The ODBC driver allocates enough memory to read the entire result set. With the entire result set from the first query stored in the result set buffer, the database connection is free to perform another query. Your client can execute this second query even though it has not processed the entire result set from the first query.

However, if you set the `ResultBufferSize` to 0, you may find that your calls to `execute()` result in the operating system killing your Perl client script. The operating system may terminate your script if the ODBC driver allocates too much memory to store a large result set.

A workaround for this behavior is limit the number of rows returned by your query. Then you can set the `ResultBufferSize` to a value that accommodates this limited result set. For example, you can estimate the amount of memory needed to store a single row of your query result. Then use the [LIMIT](#) and [OFFSET](#) clauses to get a specific number of rows that will fit into the space you allocated using `ResultBufferSize`. If the results of your query is able to fit within the limited result set buffer, you can then perform additional queries with the same database connection. This solution makes your code more complex as you will need to perform multiple queries to get the entire result set. Also, it is not appropriate in cases where you need to operate on an entire result set at once, rather than just a portion of it at a time.

A better solution is to use separate database connections for each query you want to perform. The overhead of the additional database connection is small compared to the resources needed to process large data sets.

Conversions Between Perl and Vertica Data Types

Perl is a loosely-typed programming language that does not assign specific data types to values. It converts between string and numeric values based on the operations being performed on the values. For this reason, Perl has little problem extracting most string and numeric data types from Vertica. All interval data types (DATE, TIMESTAMP, etc.) are converted to strings. You can use several different date and time handling Perl modules to manipulate these values in your scripts.

Vertica NULL values translate to Perl's undefined (undef) value. When reading data from columns that can contain NULL values, you should always test whether a value is defined before using it.

When inserting data into Vertica, Perl's DBI module attempts to coerce the data into the correct format. By default, it assumes column values are VARCHAR unless it can determine that they are some other data type. If given a string value to insert into a column that has an integer or numeric data type, DBI attempts to convert the string's contents to the correct data type. If the entire string can be converted to a value of the appropriate data type, it inserts the value into the column. If not, inserting the row of data fails.

DBI transparently converts integer values into numeric or float values when inserting into column of FLOAT, NUMERIC, or similar data types. It converts numeric or floating values to integers only when there would be no loss of precision (the value to the right of the decimal point is 0). For example, it can insert the value 3.0 into an INTEGER column since there is no loss of precision when converting the value to an integer. It cannot insert 3.1 into an INTEGER column, since that would result in a loss of precision. It returns an error instead of truncating the value to 3.

The following example demonstrates some of the conversions that the DBI module performs when inserting data into Vertica.

```
#!/usr/bin/perl
use strict;
use DBI;
# Create a hash reference that holds a hash of parameters for the
# connection.
my $attr = {AutoCommit => 0, # Turn off autocommit
            PrintError => 0 # Turn off print error. Manually handled
            };
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
    $attr);
if (defined DBI::err) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
```

```
print "Connection AutoCommit state is: " . $dbh->{AutoCommit} . "\n";
# Create table to hold inserted data
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;");
$dbh->do("CREATE TABLE TEST( \
    C_ID INT, \
    C_FP FLOAT,\
    C_VARCHAR VARCHAR(100),\
    C_DATE DATE, C_TIME TIME,\
    C_TS TIMESTAMP,\
    C_BOOL BOOL)");
# Populate an array of arrays with values.
my @data = (
    # Start with matching data types
    [1,1.111,'Matching datatypes','2001-01-01','01:01:01'
     , '2001-01-01 01:01:01','t'],
    # Force floats -> int and int -> float.
    [2.0,2,"Ints <-> floats",'2002-02-02','02:02:02'
     , '2002-02-02 02:02:02',1],
    # Float -> int *only* works when there is no loss of precision.
    # this row will fail to insert:
    [3.1,3,"float -> int with trunc?",'2003-03-03','03:03:03'
     , '2003-03-03 03:03:03',1],
    # String values are converted into numbers
    ["4","4.4","Strings -> numbers", '2004-04-04','04:04:04',
     , '2004-04-04 04:04:04',0],
    # String -> numbers only works if the entire string can be
    # converted into a number
    ["5 and a half","5.5","Strings -> numbers", '2005-05-05',
     '05:05:05', , '2005-05-05 05:05:05',0],
    # Number are converted into string values automatically,
    # assuming they fit into the column width.
    [6,6.6,3.14159, '2006-06-06','06:06:06',
     , '2006-06-06 06:06:06',0],
    # There are some variations in the accepted date strings
    [7,7.7,'Date/time formats', '07/07/2007','07:07:07',
     , '07-07-2007 07:07:07',1],
);
# Create a prepared statement to use parameters for inserting values.
my $sth = $dbh->prepare_cached("INSERT into test values(?,?,?,?,?,?)");
my $rowcount = 0; # Count # of rows
# Loop through the arrays to insert values
foreach my $tuple (@data) {
    $rowcount++;
    # Insert the row
    my $retval = $sth->execute(@$tuple);

    # See if the row was successfully inserted.
    if ($retval == 1) {
        # Value of 1 means the row was inserted (1 row was affected by insert)
        print "Row $rowcount successfully inserted\n";
    } else {
        print "Inserting row $rowcount failed with error " .
            $sth->state . " " . $sth->errstr . "\n";
    }
}
# Commit the data
$dbh->commit();
# Prepare a query to get the content of the table
$sth = $dbh->prepare("SELECT * FROM TEST ORDER BY C_ID ASC");
$sth->execute() or die "Error: " . $dbh->errstr;
```


The following example Perl script demonstrates directly inserting UTF-8 strings into Vertica and then reading them back. The example writes a text file with the output, since there are many problems displaying Unicode characters in terminal windows or consoles.

```
#!/usr/bin/perl
use strict;
use DBI;
# Open a connection using a DSN.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123");
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
# Output to a file. Displaying Unicode characters to a console or terminal
# window has many problems. This outputs a UTF-8 text file that can
# be handled by many Unicode-aware text editors:
open OUTFILE, '>:utf8', "unicodeout.txt";
# See if the DBD::ODBC driver was compiled with Unicode support. If this returns
# 1, your Perl script will get strings from the driver with the UTF-8
# flag set on them, ensuring that Perl handles them correctly.
print OUTFILE "Was DBD::ODBC compiled with Unicode support? " .
    $dbh->{odbc_has_unicode} . "\n";

# Create a table to hold VARCHARs
$dbbh->do("DROP TABLE IF EXISTS TEST CASCADE;");

# Create a table to hold data. Remember that the width of the VARCHAR column
# is the number of bytes set aside to store strings, which often does not equal
# the number of characters it can hold when it comes to Unicode!
$dbbh->do("CREATE TABLE test( C_VARCHAR VARCHAR(100) )");
print OUTFILE "Inserting data...\n";
# Use Do to perform simple inserts
$dbbh->do("INSERT INTO test VALUES('Hello')");
# This string contains several non-latin accented characters and symbols, encoded
# with Unicode escape notation. They are converted by Perl into UTF-8 characters
$dbbh->do("INSERT INTO test VALUES('My favorite band is " .
    "\N{U+00DC}m1\N{U+00E4}\N{U+00FC}t \N{U+00D6}v\N{U+00EB}rk\N{U+00EF}11" .
    " \N{U+263A}')");
# Some Chinese (Simplified) characters. This again uses escape sequence
# that Perl translates into UTF-8 characters.
$dbbh->do("INSERT INTO test VALUES('\x{4F60}\x{597D}')");
print OUTFILE "Getting data...\n";
# Prepare a query to get the content of the table
my $sth = $dbh->prepare_cached("SELECT * FROM test");
# Execute the query by calling execute on the statement handle
$sth->execute();
# Loop through result rows while we have them
while (my @row = $sth->fetchrow_array()) {
    # Loop through the column values
    foreach my $column (@row) {
        print OUTFILE "$column\t";
    }
    print OUTFILE "\n";
}
close OUTFILE;
$dbbh->disconnect();
```

Viewing the unicodeout.txt file in a UTF-8-capable text editor or viewer displays:

```
Was DBD::ODBC compiled with Unicode support? 1
Inserting data...
Getting data...
My favorite band is Ümläüt Övärkill @
你好
Hello
```

Note: Terminal windows and consoles often have problems properly displaying Unicode characters. That is why the example writes the output to a text file. With some text editors, you may need to manually set the encoding of the text file to UTF-8 in order for the characters to properly appear (and the font used to display text must have a full Unicode character set). If the character still do not show up, it may be that your version of DBD::ODBC was not compiled with UTF-8 support.

See Also

- [Unicode Character Encoding](#)
- [Required ODBC Driver Configuration Settings for Linux and UNIX](#)

Programming PHP Client Applications

You can connect to Vertica through PHP-ODBC using the [Unix ODBC](#) or [iODBC](#) library.

In order to use PHP with Vertica, you must install the following packages (and their dependencies):

- php
- php-odbc
- php-pdo
- UnixODBC (if you are using the Unix ODBC driver)
- libiodbc (if you are using the iODBC driver)

PHP on Linux

PHP is available with most Linux operating systems as a module for the Apache web server. Check your particular Linux repository for PHP RPMs or Debian packages. You can also build PHP from source. See the PHP web site for documentation and source downloads.

PHP on Windows

PHP is available for windows for both the Apache and IIS web servers. You can download PHP for Windows and view installation instructions at the PHP web site.

The PHP ODBC Drivers

PHP supports both the [UnixODBC](#) drivers and [iODBC](#) drivers. Both drivers use PHP's ODBC database abstraction layer.

Setup

You must read [Programming ODBC Client Applications](#) before connecting to Vertica through PHP. The following example ODBC configuration entries detail the typical settings required for PHP ODBC connections. The driver location assumes you have copied the Vertica drivers to `/usr/lib64`.

Example odbc.ini

```
[ODBC Data Sources]
VerticaDSNunixodbc = exampleldb
VerticaDNSiodbc = exampleldb2
[VerticaDSNunixodbc]
Description = VerticaDSN Unix ODBC driver
Driver = /usr/lib64/libverticaodbc.so
Database = Telecom
Servername = localhost
Username = dbadmin
Password =
Port = 5433
[VerticaDNSiodbc]
Description = VerticaDSN iODBC driver
Driver = /usr/lib64/libverticaodbc.so
Database = Telecom
Servername = localhost
Username = dbadmin
Password =
Port = 5433
```

Example odbcinst.ini

```
# Vertica
[VerticaDSNunixodbc]
Description = VerticaDSN Unix ODBC driver
Driver = /usr/lib64/libverticaodbc.so
[VerticaDNSiodbc]
Description = VerticaDSN iODBC driver
Driver = /usr/lib64/libverticaodbc.so
[ODBC]
Threading = 1
```

Verify the Vertica UnixODBC or iODBC Library

Verify the Vertica UnixODBC library can load all dependant libraries with the following command (assuming you have copies the libraries to /usr/lib64):

For example:

```
ldd /usr/lib64/libverticaodbc.so
```

You must resolve any "not found" libraries before continuing.

Test Your ODBC Connection

Test your ODBC connection with the following.

```
isql -v VerticaDSN
```

PHP Unicode Support

PHP does not offer native Unicode support. PHP only supports a 256-character set. However, PHP provides the UTF-8 functions `utf8_encode()` and `utf8_decode()` to provide some basic Unicode functionality.

See the PHP manual for [strings](#) for more details about PHP and Unicode.

Querying the Database Using PHP

The example script below details the use of PHP ODBC functions to connect to the Vertica Analytics Platform.

```
<?php
# Turn on error reporting
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
# A simple function to trap errors from queries
function errortrap_odbc($conn, $sql) {
    if(!$rs = odbc_exec($conn,$sql)) {
        echo "<br/>Failed to execute SQL: $sql<br/>" . odbc_errormsg($conn);
    } else {
        echo "<br/>Success: " . $sql;
    }
    return $rs;
}
```

```
}
# Connect to the Database
$dsn = "VerticaDSNunixodbc";
$conn = odbc_connect($dsn, '', '') or die ("<br/>CONNECTION ERROR");
echo "<p>Connected with DSN: $dsn</p>";
# Create a table
$sql = "CREATE TABLE TEST(
    C_ID INT,
    C_FP FLOAT,
    C_VARCHAR VARCHAR(100),
    C_DATE DATE, C_TIME TIME,
    C_TS TIMESTAMP,
    C_BOOL BOOL)";
$result = errortrap_odbc($conn, $sql);
# Insert data into the table with a standard SQL statement
$sql = "INSERT into test values(1,1.1,'abcdefg1234567890','1901-01-01','23:12:34
','1901-01-01 09:00:09','t')";
$result = errortrap_odbc($conn, $sql);
# Insert data into the table with odbc_prepare and odbc_execute
$values = array(2,2.28,'abcdefg1234567890','1901-01-01','23:12:34','1901-01-01 0
9:00:09','t');
$statement = odbc_prepare($conn,"INSERT into test values(?,?,?,?,,?,?)");
if(!$result = odbc_execute($statement, $values)) {
    echo "<br/>odbc_execute Failed!";
} else {
    echo "<br/>Success: odbc_execute.";
}
# Get the data from the table and display it
$sql = "SELECT * FROM TEST";
if($result = errortrap_odbc($conn, $sql)) {
    echo "<pre>";
    while($row = odbc_fetch_array($result) ) {
        print_r($row);
    }
    echo "</pre>";
}
# Drop the table and projection
$sql = "DROP TABLE TEST CASCADE";
$result = errortrap_odbc($conn, $sql);
# Close the ODBC connection
odbc_close($conn);
?>
```

Example Output

The following is the example output from the script.

```
Success: CREATE TABLE TEST( C_ID INT, C_FP FLOAT, C_VARCHAR VARCHAR(100), C_DATE DATE, C_TIME TIME,
C_TS TIMESTAMP, C_BOOL BOOL)
Success: INSERT into test values(1,1.1,'abcdefg1234567890','1901-01-01','23:12:34 ','1901-01-01
09:00:09','t')
Success: odbc_execute.
Success: SELECT * FROM TEST
Array
(
    [C_ID] => 1
```

```
[C_FP] => 1.1
[C_VARCHAR] => abcdefg1234567890
[C_DATE] => 1901-01-01
[C_TIME] => 23:12:34
[C_TS] => 1901-01-01 09:00:09
[C_BOOL] => 1
)
Array
(
  [C_ID] => 2
  [C_FP] => 2.28
  [C_VARCHAR] => abcdefg1234567890
  [C_DATE] => 1901-01-01
  [C_TIME] => 23:12:34
  [C_TS] => 1901-01-01 23:12:34
  [C_BOOL] => 1
)
Success: DROP TABLE TEST CASCADE
```

Managing Query Execution Between the Client and Vertica

Multiple Active Result Sets (MARS)

ResultBufferSize

ResultBufferSize

By default, Vertica uses the `ResultBufferSize` parameter to determine the maximum size (in bytes) of a result set that a client can retrieve from a server. When `ResultBufferSize` is enabled, Vertica sends rows of data directly to the client making the query. The number of rows returned to the client at each fetch of data depends on the size (in bytes) of the `ResultBufferSize` parameter.

Sometimes, the size of the result set requested by the client is greater than what the `ResultBufferSize` parameter allows. In such cases, Vertica retrieves only a portion of the result set at a time. Each fetch of data returns the amount of data equal to the size set by the `ResultBufferSize` parameter. Ultimately, as the client iterates over the individual fetches of data, the entire result set is returned.

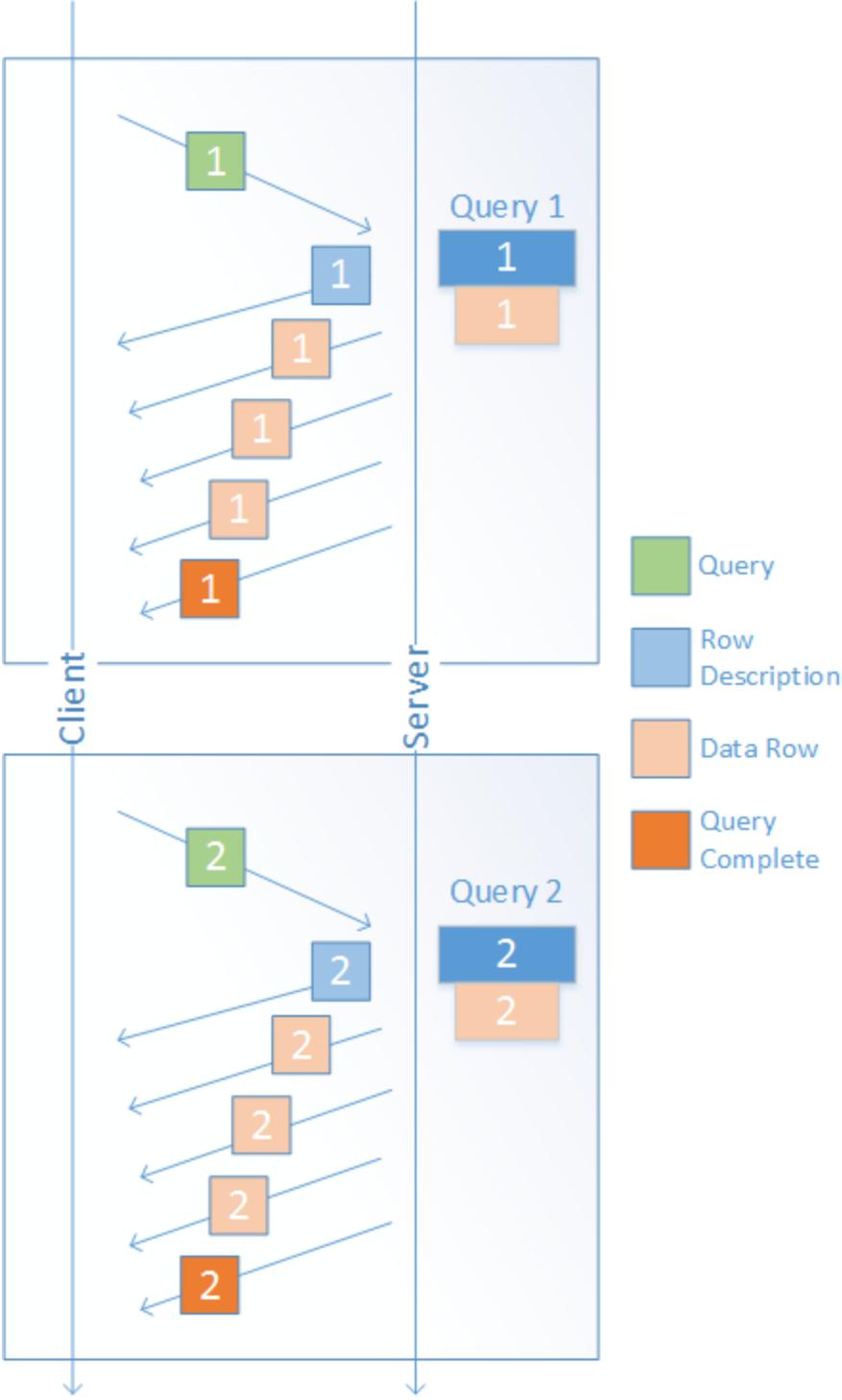
Benefits of ResultBufferSize

If you are concerned with the effect of your queries on network latency, `ResultBufferSize` may provide an advantage over MARS. MARS requires that the client wait until all rows of data are written to the server before the client can retrieve the data. This delay may cause latency issues for your network while waiting for the results to be stored.

In addition, MARS requires that you send two separate requests to return rows of data. The first request performs the query execution which stores the result set on the server. The second request retrieves the data rows that are stored on the server. With `ResultBufferSize`, you only need to send one request. This request both executes and retrieves the data rows of interest.

Query Execution with ResultBufferSize

The following graphic shows how Vertica returns rows of data from a database to the client with `ResultBufferSize` enabled:



The query execution performs the following steps:

1. The client sends a query, such as a [SELECT](#) statement, to the server. In the preceding graphic, the first query is named Query 1.

2. The server receives the client's request and begins to send both a description of the result set and the requested rows of data back to the client.
3. After all possible rows are returned to the client, the execution is complete. The size of the data set returned equals either that of the data that was requested or the maximum amount of data that ResultBufferSize parameter can retrieve. If the ResultBufferSize maximum size is not yet reached, Vertica can execute Query 2.

The server can accept Query 2 and perform the same steps that it did for Query 1. If the results for Query 1 had reached the maximum ResultBufferSize allowable, Vertica could not execute Query 2 until the client freed the results from Query 1.

After Query 2 runs, you cannot view the results you retrieved for Query 1, unless you execute Query 1 again.

Setting an Unlimited Buffer Size

Setting ResultBufferSize to 0 tells the client driver to use an unlimited result set buffer. With this setting, the client library allocates as much memory as it needs to read the entire result set of a query. You may choose to set ResultBufferSize to 0 if you want to simulate having multiple active queries over a single database connection at the same time. With an unlimited buffer size, your client can run a query and have its entire result set stored in memory. This ends the first query, so your client can execute a second query before it fully processes the results of the first query.

A drawback of this method is that your query may consume too much memory if your queries return large result sets. This over-allocation of memory can result in the operating system terminating your client. Due to this risk, consider using multiple database connections instead of trying to reuse a single connection for multiple queries. The overhead of multiple database connections is small compared to the overall amount of resources required to process a large data set.

Multiple Active Result Sets (MARS)

You can only enable MARS when you connect to Vertica using a JDBC client connection. MARS allows the execution of multiple queries on a single connection. While ResultBufferSize sends the results of a query directly to the client, MARS stores the results first on the server. Once query execution has finished and all of the results have been stored, you can make a retrieval request to the server to have rows returned to the client.

MARS is set at the session level and must be enabled for every new session. When MARS is enabled, `ResultBufferSize` is disabled. No error is returned, however the `ResultBufferSize` parameter is ignored.

Benefits of MARS

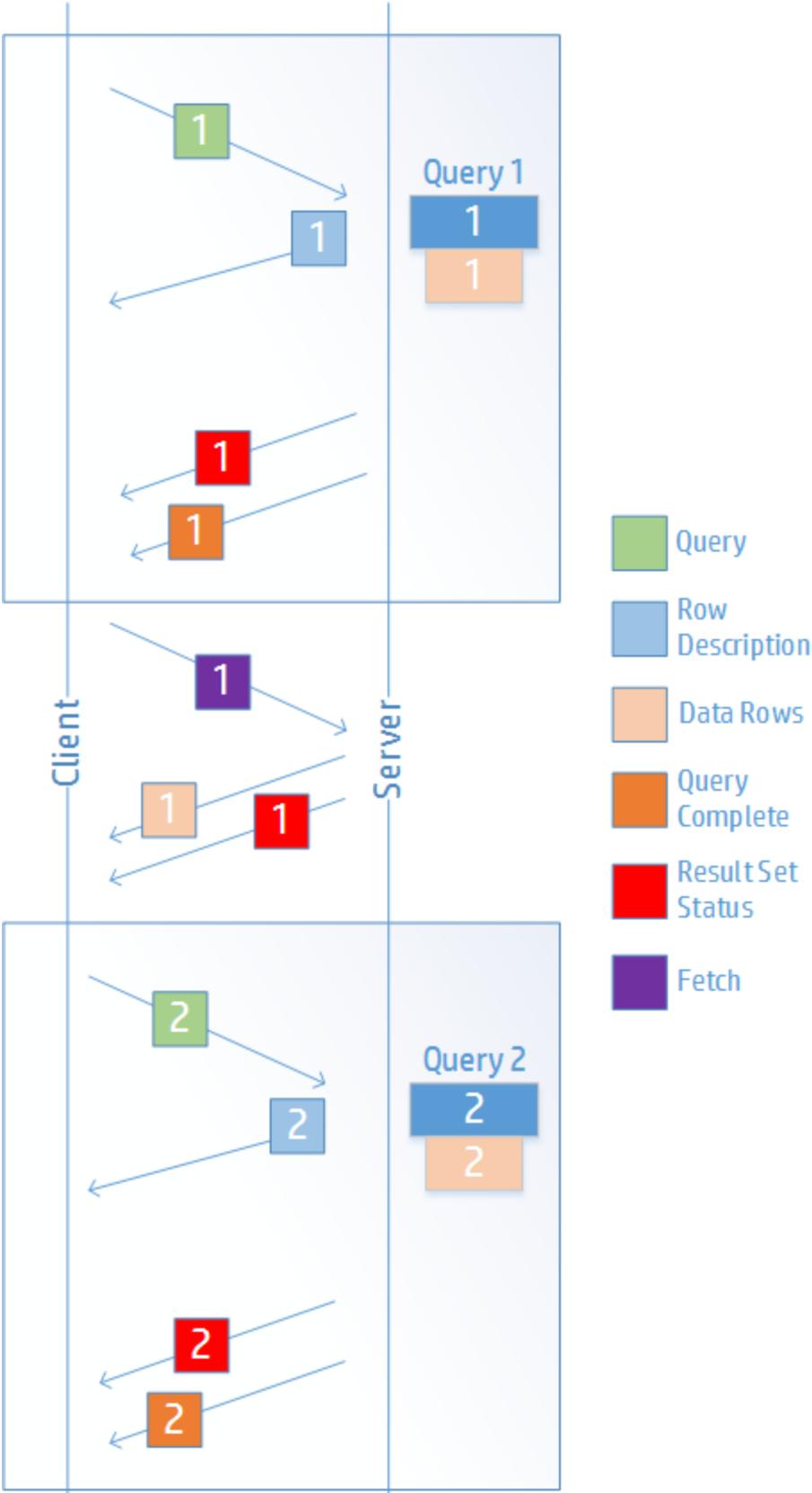
In comparison with `ResultBufferSize`, MARS enables you to store multiple result sets from different queries at the same time. You can also send new queries before all of the results of a previous result set have been returned to the client. This allows applications to decouple query execution from result retrieval so that, on a single connection, you can process different results at the same time.

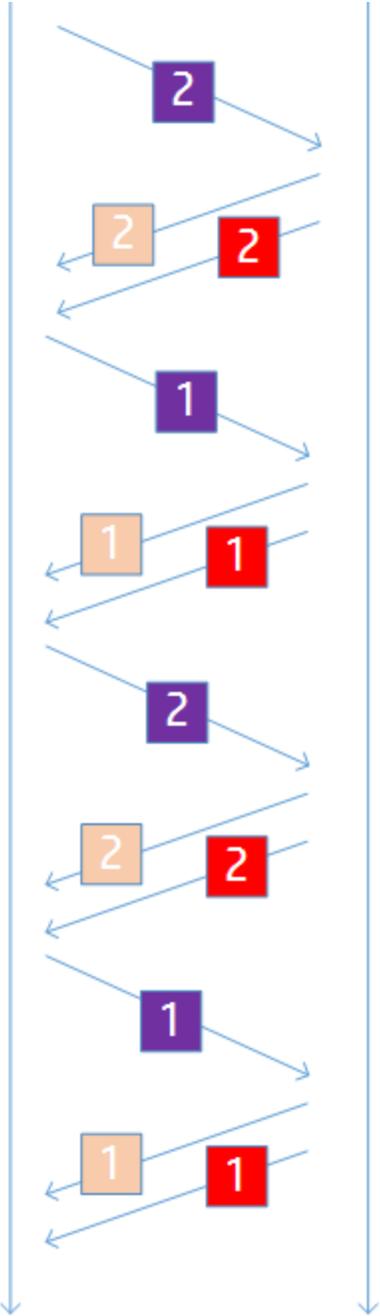
When you enable `ResultBufferSize`, you must wait until all result sets have been returned to the client before a new query can be executed.

Another benefit of MARS is that it allows you to free up query resources faster than `ResultBufferSize` allows. While a query is running, resources are held by that query session. When `ResultBufferSize` is enabled, a client that is performing slowly might read a single row of a result set and then have to stop to retrieve the next row. This prevents the query from finishing quickly and, therefore, prevents the resources used from being freed up for other applications. With MARS, the speed of the client is irrelevant to the reading of rows. As soon as the results are written to the MARS storage, the resources are freed and the speed at which the client retrieves rows no longer matters.

Query Execution with MARS

The following graphic demonstrates how multiple queries to the server are handled when MARS is enabled:





Query 1:

- 1. Query 1 is sent to the server.
- 2. Query 1's row description and the status of its result set are returned to the client. However, no results are returned to the client at this time.
- 3. Query 1 completes and its results are saved on the server.

- a. You can now send commands to retrieve the rows of Query 1's result set. These rows are stored on the server. Retrieved rows are sent to the client along with the status of the result set. By keeping track of the status of the result set, Vertica is able to keep track of which rows have been retrieved from the server.
4. Now that Query 1 has successfully completed, and its result sets are being stored on the server, Query 2 can be executed.

Query 2:

1. Query 2 is sent to the server.
2. Query 2's row description and the status of its result set are returned to the client. However, no results are returned to the client at this time.
3. Query 2 completes and its results are stored on the server. Both Query 1 and Query 2 now have result sets stored on the server.
4. You can now send retrieval requests to both Query 1 and Query 2's result sets that are stored on the server. Whenever a retrieval request is made for rows from Query 1, the request is sent and rows and the result set status are sent to the client. The same occurs for Query 2.

Once all rows have been read by the client, the MARS storage on the server closes the active results session. The MARS storage on the server is then freed to store more data. The MARS storage also closes and frees once your session is finished.

Enabling and Disabling MARS

You can enable and disable MARS in two different ways:

1. To enable MARS using the JDBC client connection properties, see [JDBC Connection Properties](#).
2. To enable MARS using the SET SESSION command, see [SET SESSION MULTIPLEACTIVERESULTSETS](#).

See Also

- [SESSION_MARS_STORE](#)
- [CLOSE_RESULTSET](#)

- [CLOSE_ALL_RESULTSETS](#)

Management API

The Management API is a REST API that you can use to view and manage Vertica databases with scripts or applications that accept REST and JSON. The response format for all requests is JSON.

cURL

cURL is a command-line tool and application library used to transfer data to or from a server. It uses URL syntax, such as HTTP and HTTPS. All API requests sent to a Vertica server must be made using HTTPS. A request made using HTTP will fail.

There are four HTTP requests that can be passed using cURL to call API methods:

Request	Description
GET	Retrieves data.
PUT	Updates data.
POST	Creates new data.
DELETE	Deletes data.

Syntax

```
curl https://<NODE>:5444/
```

Options

-h --help	Lists all available command-line options.
-H --header	Allows you to use custom headers in your command. This is useful for sending a request that requires a VerticaAPIKEY. <pre>curl -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/</pre>

<code>-k --insecure</code>	<p>Allows SSL connections without certificate validation.</p> <pre>curl -k https://<NODE>:5444/</pre>
<code>-X --request</code>	<p>Specifies the custom request used when communicating with a server.</p> <pre>curl -X REQUEST https://<NODE>:5444/</pre> <p>Can be one of the following values:</p> <ul style="list-style-type: none">• GET• PUT• POST• DELETE <p>Note: If no request is specified, cURL automatically defaults to the GET request.</p>

There are many more options available to add to your cURL query. For a comprehensive list with descriptions, visit the [cURL Documentation Website](#).

General API Information

These API calls can interact with either standard Vertica nodes or Management Console nodes.

GET /	Returns the agent-specific information useful for version checking and service discovery.
GET api	Returns a list of api objects and properties.

GET /

Returns API version information and a list of links to child resources for the Management API.

Resource URL

```
https://<NODE>:5444/
```

Authentication

Not required.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/</pre>
------------	---------------------------------------

Response:

```
{
  "body": {
    "mime-types": [
      "default",
      "application/vertica.database.configuration.json-v2",
      "application/json",
      "application/vertica.nodes.json-v2",
      "default",
      "application/json",
      "default",
      "application/json",
      "application/vertica.jobs.json-v2",
      "default",
      "application/vertica.hosts.json-v2",
      "application/json",
      "default",
      "application/vertica.hosts.json-v2",
      "application/json",
      "default",
    ]
  }
}
```

```
    "application/json",
    "application/vertica.host.json-v2",
    "default",
    "application/vertica.hosts.json-v2",
    "application/json",
    "application/vertica.nodes.json-v2",
    "default",
    "application/json",
    "default",
    "application/json",
    "application/vertica.database.json-v2",
    "default",
    "application/vertica.hosts.json-v2",
    "application/json",
    "default",
    "application/vertica.hosts.json-v2",
    "application/json",
    "default",
    "application/json",
    "application/vertica.databases.json-v2",
    "application/vertica.nodes.json-v2",
    "default",
    "application/json",
    "application/vertica.agent.json-v2",
    "default",
    "application/json",
    "default",
    "application/vertica.users.json-v2",
    "application/json"
  ],
  "version": "7.1.0"
},
"href": "/",
"links": [
  "/databases",
  "/hosts",
  "/nodes",
  "/licenses",
  "/webhooks",
  "/backups",
  "/restore",
  "/jobs"
],
"mime-type": "application/vertica.agent.json-v2"
}
```

GET api

Displays a list of all Management API commands and a brief description for each command. You can also access an HTML version of this information at <https://<NODE>:5444/api.html>. Replace <NODE> with the URL of a cluster node.

Resource URL

```
https://<NODE>:5444/api
```

Authentication

Not required.

Parameters

None.

Example Request

GET	<code>https://<NODE>:5444/api</code>
------------	--

Response:

```
[
  {
    "accepts": {},
    "description": " Returns the agent specific information useful for version checking and
service discovery ",
    "method": "GET",
    "params": [],
    "route": "/"
  },
  {
    "accepts": {},
    "description": " build the list of cluster objects and properties and return it as a JSON
formatted array ",
    "method": "GET",
    "params": [],
    "route": "/api"
  }
]
```

```
    },
    {
      "accepts": {},
      "description": " list all the backups that have been created for all vbr configuration files
( *.ini ) that are located in the /opt/vertica/config directory. ",
      "method": "GET",
      "params": [],
      "route": "/backups"
    },
    {
      "accepts": {},
      "description": " create a new backup as defined by the given vbr configuration script base
(filename minus the .ini extenstion) ",
      "method": "POST",
      "params": [],
      "route": "/backups/:config_script_base"
    },
    {
      "accepts": {},
      "description": " get the detail for a specific backup archive ",
      "method": "GET",
      "params": [],
      "route": "/backups/:config_script_base/:archive_id"
    },
    {
      "accepts": {},
      "description": " delete a backup based on the config ini file script",
      "method": "DELETE",
      "params": [],
      "route": "/backups/:config_script_base/:backup_id"
    },
    {
      "accepts": {},
      "description": " build the list of databases, their properties, and current status ( from
cache ) and return it as a JSON formatted array ",
      "method": "GET",
      "params": [],
      "route": "/databases"
    },
    {
      "accepts": {},
      "description": " Create a new database by supplying a valid set of parameters ",
      "method": "POST",
      "params": [
        "name      : name of the database to create",
        "passwd   : password used by the database administrative user",
        "only     : optional list of hostnames to include in database",
        "exclude  : optional list of hostnames to exclude from the database",
        "catalog  : directory used for the vertica catalog",
        "data     : directory used for the initial vertica storage location",
        "port     : port the database will listen on (default 5433)"
      ],
      "route": "/databases"
    },
    {
      "accepts": {},
      "description": " Retrieve the database properties structure ",
      "method": "GET",
      "params": [],
      "route": "/databases/:database_name"
    }
  ],
  {
    "accepts": {},
    "description": " Retrieve the database properties structure ",
    "method": "GET",
    "params": [],
    "route": "/databases/:database_name"
  }
],
{
  "accepts": {},
  "description": " Retrieve the database properties structure ",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name"
}
```

```
  },
  {
    "accepts": {},
    "description": " Control / alter a database values using the PUT http method ",
    "method": "PUT",
    "params": [
      "action : value one of start|stop|rebalance|wla"
    ],
    "route": "/databases/:database_name"
  },
  {
    "accepts": {},
    "description": " Delete an existing database ",
    "method": "DELETE",
    "params": [],
    "route": "/databases/:database_name"
  },
  {
    "accepts": {},
    "description": " retrieve the current parameters from the database. if its running return 503 Service Unavailable ",
    "method": "GET",
    "params": [
      "user_id : vertica database username",
      "passwd : vertica database password"
    ],
    "route": "/databases/:database_name/configuration"
  },
  {
    "accepts": {},
    "description": " set a list of parameters in the database. if its not running return 503 Service Unavailable ",
    "method": "PUT",
    "params": [
      "user_id : vertica database username",
      "passwd : vertica database password",
      "parameter : value vertica parameter/key combo"
    ],
    "route": "/databases/:database_name/configuration"
  },
  {
    "accepts": {},
    "description": " list the hosts that are current used by this database",
    "method": "GET",
    "params": [],
    "route": "/databases/:database_name/hosts"
  },
  {
    "accepts": {},
    "description": " add a host to the database ",
    "method": "POST",
    "params": [
      "user_id : vertica database username",
      "passwd : vertica database password"
    ],
    "route": "/databases/:database_name/hosts"
  },
  {
    "accepts": {},
    "description": " remove a host from the database",
```

```
    "method": "DELETE",
    "params": [],
    "route": "/databases/:database_name/hosts/:host_id"
  },
  {
    "accepts": {},
    "description": " start the database process on a specific host participating in this
database.",
    "method": "POST",
    "params": [],
    "route": "/databases/:database_name/hosts/:host_id/process"
  },
  {
    "accepts": {},
    "description": " run the stop action against the given database for a specific host",
    "method": "DELETE",
    "params": [],
    "route": "/databases/:database_name/hosts/:host_id/process"
  },
  {
    "accepts": {},
    "description": " replace a host with a standby host in the database ",
    "method": "POST",
    "params": [
      "user_id : vertica database username",
      "passwd : vertica database password"
    ],
    "route": "/databases/:database_name/hosts/:host_id/replace_with/:host_id_new"
  },
  {
    "accepts": {},
    "description": " return the vertica license that this database is using ",
    "method": "GET",
    "params": [
      "user_id : vertica database user",
      "passwd : vertica databse password"
    ],
    "route": "/databases/:database_name/license"
  },
  {
    "accepts": {},
    "description": " this method upgrades the license in the database by using the license in
/opt/vertica/config/share ",
    "method": "PUT",
    "params": [
      "user_id : vertica database user",
      "passwd : vertica databse password"
    ],
    "route": "/databases/:database_name/license"
  },
  {
    "accepts": {},
    "description": " return all the feature licenses that this database is using ",
    "method": "GET",
    "params": [
      "user_id : vertica database user",
      "passwd : vertica databse password"
    ],
    "route": "/databases/:database_name/licenses"
  },
}
```

```
{
  "accepts": {},
  "description": " build the list of nodes for a given database, their properties, and current
status ( from cache ) and return it as a JSON formatted array ",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/nodes"
},
{
  "accepts": {},
  "description": " build the list of nodes for a given database, their properties, and current
status and return it as a JSON formatted array ",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/nodes/:node_id"
},
{
  "accepts": {},
  "description": " run the start action against the given database ",
  "method": "POST",
  "params": [
    "epoch  : start the database from this epoch",
    "include : start the database on these hosts only"
  ],
  "route": "/databases/:database_name/process"
},
{
  "accepts": {},
  "description": " easy way to see if a database is running -- returns state of UP or DOWN",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/process"
},
{
  "accepts": {},
  "description": " run the stop action against the given database ",
  "method": "DELETE",
  "params": [
    "user_id : vertica database username",
    "passwd  : vertica database password"
  ],
  "route": "/databases/:database_name/process"
},
{
  "accepts": {},
  "description": " run the rebalance data action against the given database - could be very
long running! ",
  "method": "POST",
  "params": [
    "user_id : vertica database username",
    "passwd  : vertica database password"
  ],
  "route": "/databases/:database_name/rebalance/process"
},
{
  "accepts": {},
  "description": " Retrieve the database properties structure ",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/status"
```

```
    },
    {
      "accepts": {},
      "description": " run the analyze workload action against the given database - could be very
long running! ",
      "method": "POST",
      "params": [
        "user_id : vertica database username",
        "passwd : vertica database password"
      ],
      "route": "/databases/:database_name/wla/process"
    },
    {
      "accepts": {},
      "description": " build a list of nodes in the cluster independent of their database
associations ",
      "method": "GET",
      "params": [],
      "route": "/hosts"
    },
    {
      "accepts": {},
      "description": " lists the properties of a given host in the cluster by calling the RESTful
service on that agent. ",
      "method": "GET",
      "params": [],
      "route": "/hosts/:hostid"
    },
    {
      "accepts": {},
      "description": " Returns a list of jobs the agent is tracking along with their current
status and exit codes ",
      "method": "GET",
      "params": [],
      "route": "/jobs"
    },
    {
      "accepts": {},
      "description": " Deletes a specific job by canceling any outstanding activity associated
with it. ",
      "method": "DELETE",
      "params": [],
      "route": "/jobs/:id"
    },
    {
      "accepts": {},
      "description": " Returns the details (the saved output) for a specific job ",
      "method": "GET",
      "params": [],
      "route": "/jobs/:id"
    },
    {
      "accepts": {},
      "description": " POST your vertica license to this url using HTTP file upload format. ",
      "method": "POST",
      "params": [
        "license : the file to upload (use html form post format)"
      ],
      "route": "/licenses"
    },
  ],
}
```

```
{
  "accepts": {},
  "description": " list the license fiel that will be used by databases created on this
cluster found in /opt/vertica/config/share/license.key ",
  "method": "GET",
  "params": [],
  "route": "/licenses"
},
{
  "accepts": {},
  "description": " build a list of nodes in the topology independent of their database
associations ",
  "method": "GET",
  "params": [],
  "route": "/nodes"
},
{
  "accepts": {},
  "description": " build a list of nodes in the topology independent of their database
associations ",
  "method": "GET",
  "params": [],
  "route": "/nodes/:nodeid"
},
{
  "accepts": {},
  "description": " restore a backup given then archive id as defined via the 'GET' method ",
  "method": "POST",
  "params": [],
  "route": "/restore/:archive_id"
},
{
  "accepts": {},
  "description": " delete a subscription from this agent. ",
  "method": "DELETE",
  "params": [],
  "route": "/webhooks/:subscriber_id"
},
{
  "accepts": {},
  "description": " post a request with a callback url to subscribe to events from this agent.
Returns a subscription_id that can be used to unsubscribe from the service. @returns subscription_
id ",
  "method": "POST",
  "params": [
    "url : full url to the callback resource"
  ],
  "route": "/webhooks/subscribe"
}
]
```

Rest APIs for the Agent

These API calls interact with standard Vertica nodes.

Backup and Restore

GET backups	Returns all the backups that have been created for all vbr configuration files (*.ini) that are located in the /opt/vertica/config directory.
POST backups/:config_script_base	Creates a new backup as defined by the given vbr configuration script base (filename without the .ini extension).
GET backups/:config_script_base/:archive_id	Returns details for a specific backup archive.
POST restore/:archive_id	Restores a backup.

Databases

GETdatabases	Returns a list of databases, their properties, and current status.
POST databases	Creates a new database by supplying a valid set of parameters.
GET databases/:database_name	Returns details about a specific database.
PUT databases/:database_name	Starts, stops, rebalances, or runs Workload Analyzer on a database.
DELETE databases/:database_name	Deletes an existing database.
GET databases/:database_name/configuration	Returns the current configuration parameters from the database.
PUT databases/:database_name/configuration	Sets one or more configuration parameters in the database.
GET databases/:database_name/hosts	Returns hosts details for a specific database.

POST databases/:database_name/hosts	Adds a new host to the database.
DELETE databases/:database_name/hosts/:host_id	Removes a host from the database.
POST databases/:database_name/hosts/:host_id/process	Starts the database process on a specific host.
DELETE databases/:database_name/hosts/:host_id/process	Stops the database on a specific host.
POST databases/:database_name/hosts/:host_id/replace_with/:host_id_new	Replaces a host with a standby host in the database.
GET databases/:database_name/license	Returns the Vertica license that the specified database is using.
PUT databases/:database_name/license	Upgrades the license in the database.
GET databases/:database_name/licenses	Returns all the feature licenses that the specified database is using.
GET databases/:database_name/nodes	Returns a list of nodes for the specified database.
GET databases/:database_name/nodes/:node_id	Returns details on a specific node for the specified database.
POST databases/:database_name/process	Starts the specified database.
GET databases/:database_name/process	Returns the state of the database as either UP or DOWN.
DELETE databases/:database_name/process	Stops the specified database on all hosts.
POST databases/:database_name/rebalance/process	Rebalances the specified database. This option can have a long run time.
POST databases/:database_name/wla/process	Runs the analyze workload action against the specified database. This option can have a long run time.

Hosts

GET hosts	Returns a list of hosts in this cluster.
GET hosts/:hostid	Returns details for a specific host in this cluster.

Jobs

GET jobs	Returns a list of jobs the agent is tracking, along with their current status and exit codes.
GET jobs/:id	Returns the details (the saved output) for a specific job.

Licenses

POST licenses	Uploads and applies a new license to this cluster.
GET licenses	Returns the license field that databases created on this cluster use.

Nodes

GET nodes	Returns a list of nodes in this cluster.
GET nodes/:nodeid	Returns details for a specific node in this cluster.

Webhooks

GET webhooks	Returns a list of active webhooks.
POST webhooks/subscribe	Creates a new webhook.
DELETE webhooks/:subscriber_id	Deletes an existing webhook.

VerticaAPIKey

The Management API requires an authentication key, named VerticaAPIKEY, to access some API resources. You can manage API keys by using the **apikeymgr** command-line tool.

```
usage: apikeymgr [-h] [--user REQUESTOR] [--app APPLICATION] [--delete]
                [--create] [--update] [--migrate]
                [--secure {restricted,normal,admin}] [--list]
```

API key management tool

optional arguments:

```
-h, --help          show this help message and exit
--user REQUESTOR   The name of the person requesting the key
--app APPLICATION  The name of the application that will use the key
--delete           Delete the key for the given R & A
--create           Create a key for the given R & A
--update          Update a key for the given R & A
--migrate          migrate the keyset to the latest format
--secure {restricted,normal,admin}
                  Set the keys security level
--list            List all the keys known
```

Example Request

To create a new VerticaAPIKEY for the dbadmin user with admin access, enter the following:

```
$ apikeymgr --user dbadmin --app vertica --create --secure admin
```

Response:

```
Requestor : dbadmin
Application: vertica
API Key   : ValidAPIKey
Synchronizing cluster...
```

Backup and Restore

You can use these API calls to perform backup and restore tasks for your database.

GET backups

Returns all the backups that have been created for all vbr configuration files (*.ini) that are located in the /opt/vertica/config directory.

POST backups/:config_script_base	Creates a new backup as defined by the given vbr configuration script base (filename without the .ini extension).
GET backups/:config_script_base/:archive_id	Returns details for a specific backup archive.
POST restore/:archive_id	Restores a backup.

GET backups

Returns a list of all backups created for vbr configuration (*.ini) files that reside in /opt/vertica/config and provides details about each backup.

Resource URL

```
https://<NODE>:5444/backups
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/backups</pre>
------------	--

Response:

```
[
  {
    "backups": [
      {
        "backup": "backup3_20140707_114852",
        "epoch": "61",
        "hosts": "v_vmart_node0001(10.20.100.247)",
        "objects": ""
      }
    ],
    "config_file": "/opt/vertica/config/backup3.ini",
    "num_backups": 1
  },
  {
    "backups": [
      {
        "backup": "backup_20140707_113737",
        "epoch": "61",
        "hosts": "v_vmart_node0001(10.20.100.247)",
```

```
        "objects": ""
      },
      {
        "backup": "backup_archive20140707_113645",
        "epoch": "60",
        "hosts": "v_vmart_node0001(10.20.100.247)",
        "objects": ""
      }
    ],
    "config_file": "/opt/vertica/config/backup.ini",
    "num_backups": 2
  }
]
```

POST backups/:config_script_base

Creates a new backup job for the backup defined in the vbr configuration script `:config_script_base`. The vbr configuration script must reside in `/opt/vertica/configuration`. The `:config_script_base` value does not include the `.ini` filename extension.

To determine valid `:config_script_base` values, see [GET backups](#).

Returns a job ID that you can use to determine the status of the job.

Resource URL

```
https://<NODE>:5444/backups/:config_script_base
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

POST	<pre>https://<NODE>:5444/backups/backup3</pre>
-------------	--

Response:

```
{
  "id": "CreateBackup-VMart-1404750602.03",
  "url": "/jobs/CreateBackup-VMart-1404750602.03"
}
```

GET backups/:config_script_base/:archive_id

Returns details on a specific backup. You must provide the `:config_script_base`. This value is the name of a vbr config file (without the `.ini` filename extension) that resides in `/opt/vertica/config`. The `:archive_id` is the value of the `backup` field that the [GET backups](#) command returns.

Resource URL

```
https://<NODE>:5444/backups/:config_script_base/:archive_id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/backups/backup3/backup3_20140707_123254</pre>
------------	--

Response:

```
{
  "backup": "backup3_20140707_123254",
  "config_file": "/opt/vertica/config/backup3.ini",
  "epoch": "62",
  "hosts": "v_vmart_node0001(10.20.100.247)",
  "objects": ""
}
```

POST restore/:archive_id

Creates a new restore job to restore the database from the backup archive identified by :archive_id. The :archive_id is the value of a *backup* field that the [GET backups](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/restore/:archive_id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

POST	<pre>https://<NODE>:5444/restore/backup3_20140707_132904</pre>
-------------	--

Response:

```
{
  "id": "RestoreBackup-VMart-1404760113.71",
  "url": "/jobs/RestoreBackup-VMart-1404760113.71"
}
```

Databases

You can use these API calls to interact with your database.

GETdatabases	Returns a list of databases, their properties, and current status.
------------------------------	--

POST databases	Creates a new database by supplying a valid set of parameters.
GET databases/:database_name	Returns details about a specific database.
PUT databases/:database_name	Starts, stops, rebalances, or runs Workload Analyzer on a database.
DELETE databases/:database_name	Deletes an existing database.
GET databases/:database_name/configuration	Returns the current configuration parameters from the database.
PUT databases/:database_name/configuration	Sets one or more configuration parameters in the database.
GET databases/:database_name/hosts	Returns hosts details for a specific database.
POST databases/:database_name/hosts	Adds a new host to the database.
DELETE databases/:database_name/hosts/:host_id	Removes a host from the database.
POST databases/:database_name/hosts/:host_id/process	Starts the database process on a specific host.
DELETE databases/:database_name/hosts/:host_id/process	Stops the database on a specific host.
POST databases/:database_name/hosts/:host_id/replace_with/:host_id_new	Replaces a host with a standby host in the database.
GET databases/:database_name/license	Returns the Vertica license that the specified database is using.
PUT databases/:database_name/license	Upgrades the license in the database.
GET databases/:database_name/licenses	Returns all the feature licenses that the specified database is using.
GET databases/:database_name/nodes	Returns a list of nodes for the specified database.
GET databases/:database_name/nodes/:node_id	Returns details on a specific node for the specified database.

POST databases/:database_name/process	Starts the specified database.
GET databases/:database_name/process	Returns the state of the database as either UP or DOWN.
DELETE databases/:database_name/process	Stops the specified database on all hosts.
POST databases/:database_name/rebalance/process	Rebalances the specified database. This option can have a long run time.
POST databases/:database_name/wla/process	Runs the analyze workload action against the specified database. This option can have a long run time.

GETdatabases

Returns a list of databases, their current status, and database properties.

Resource URL

```
https://<NODE>:5444/databases
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/databases</pre>
------------	--

An example of the full request using cURL:

```
curl -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/databases
```

Response:

```
{
  "body": [
    {
      "href": "/databases/VMart",
      "mime-type": [
        "application/vertica.database.json-v2"
      ],
      "name": "VMart",
      "port": "5433",
      "status": "UP"
    },
    {
      "href": "/databases/testDB",
      "mime-type": [
        "application/vertica.database.json-v2"
      ],

```

```
        "name": "testDB",  
        "port": "5433",  
        "status": "DOWN"  
    }  
],  
"href": "/databases",  
"links": [  
    "[:database_name"  
],  
"mime-type": "application/vertica.databases.json-v2"  
}
```

POST databases

Creates a job to create a new database with the provided parameters.

Important: You must stop any running databases on the nodes on which you want to create the new database. If you do not, database creation fails.

Returns a job ID that can be used to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/database
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

name	Name of the database to create.
passwd	Password for the new database.
only	Optional list of hostnames to include in the database. By default, all nodes in the cluster are added to the database.
exclude	Optional list of hostnames to exclude from the database.
catalog	Path of the catalog directory.
data	Path of the data directory.
port	Port where the database listens for client connections. Default is 5433.

Example Request

POST	<pre>https://<NODE>:5444/databases?passwd=vertica&name=testDB &catalog=%2Fhome%2Fdbadmin%2FtestDB &data=%2Fhome%2Fdbadmin%2FtestDB</pre>
-------------	--

Response:

```
{  
  "jobid": "CreateDatabase-testDB-2014-07-07 15:49:53.219445",  
  "resource": "/jobs/CreateDatabase-testDB-2014-07-07 15:49:53.219445",  
  "userid": "dbadmin"  
}
```

GET databases/:database_name

Returns details about a specific database. The `:database_name` is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/databases/:database_name
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/databases/VMart</pre>
------------	--

Response:

```
{
  "body": {
    "database_id": "VMart",
    "id": "VMart",
    "nodes": "v_vmart_node0001,v_vmart_node0002,v_vmart_node0003",
    "nodes_new": [
      {
        "catalog_base": "/home/dbadmin",
        "data_base": "/home/dbadmin",
        "host": "10.20.100.247",
        "id": "v_vmart_node0001"
      },
      {
        "catalog_base": "/home/dbadmin",
        "data_base": "/home/dbadmin",
        "host": "10.20.100.248",
        "id": "v_vmart_node0002"
      },
      {

```

```
        "catalog_base": "/home/dbadmin",
        "data_base": "/home/dbadmin",
        "host": "10.20.100.249",
        "id": "v_vmart_node0003"
      }
    ],
    "path": "/home/dbadmin/VMart",
    "port": "5433",
    "restartpolicy": "ksafe",
    "status": "UP"
  },
  "href": "/databases/VMart",
  "links": [
    "/configuration",
    "/hosts",
    "/license",
    "/nodes",
    "/process",
    "/rebalance/process",
    "/status",
    "/wla/process"
  ],
  "mime-type": "application/vertica.database.json-v2"
}
```

PUT databases/:database_name

Creates a job to run the action specified by the *action* parameter against the database identified by *:database_name*. The *:database_name* is the value of the *name* field that the [GETdatabases](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **normal** level security or higher.

Parameters

user_id	A database username.
passwd	A password for the username.
action	Can be one of the following values: <ul style="list-style-type: none">• start — Start the database.• stop — Stop the database.• rebalance — Rebalance the database.• wla — Run Work Load Analyzer against the database.

Example Request

PUT	<pre>https://<NODE>:5444/databases/testDB?user_id=dbadmin"&"passwd=vertica"&"action=stop</pre>
-----	--

Response:

```
{  
  "id": "StopDatabase-testDB-2014-07-20 13:28:49.321744",  
  "url": "/jobs/StopDatabase-testDB-2014-07-20 13:28:49.321744"  
}
```

DELETE databases/:database_name

Creates a job to delete (drop) an existing database on the cluster. To perform this operation, you must first stop the database. The `:database_name` is the value of the `name` field that the [GETdatabases](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

None.

Example Request

DELETE	<pre>https://<NODE>:5444/databases/TestDB</pre>
---------------	---

Response:

```
{
  "id": "DropDatabase-TestDB-2014-07-18 12:50:33.332383",
  "url": "/jobs/DropDatabase-TestDB-2014-07-18 12:50:33.332383"
}
```

GET databases/:database_name/configuration

Returns a list of configuration parameters for the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/databases/:database_name/configuration
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

user_id	A database username.
passwd	The password for the username.

Example Request

GET	<pre>https://<NODE>:5444/databases/testDB/configuration</pre>
------------	---

Response:

This API call returns over 100 configuration parameters.. The following response is a small subset of the total amount returned.

```
[
  {
    "catalog_value": "1",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "1",
    "database_value": "1",
    "default_value": "1",
    "description": "No of active partitions",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
```

```
    "parameter_name": "ActivePartitionCount",
    "source": "DEFAULT"
  },
  {
    "catalog_value": "180",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "180",
    "database_value": "180",
    "default_value": "180",
    "description": "Interval between advancing the AHM (seconds)",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
    "parameter_name": "AdvanceAHMInterval",
    "source": "DEFAULT"
  },
  {
    "catalog_value": "3",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "3",
    "database_value": "3",
    "default_value": "3",
    "description": "HDFS replication factor for Vertica data",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
    "parameter_name": "HadoopFSReplication",
    "source": "DEFAULT"
  },
  {
    "catalog_value": "",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "",
    "database_value": "",
    "default_value": "",
    "description": "Path to the java binary for executing UDx written in Java",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
    "parameter_name": "JavaBinaryForUDx",
    "source": "DEFAULT"
  },
  {
    "catalog_value": "80",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "80",
    "database_value": "80",
    "default_value": "80",
    "description": "The max amount of memory TopK(Heap) can use in MB",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
    "parameter_name": "TopKHeapMaxMem",
    "source": "DEFAULT"
  },
  {
```

```
    "catalog_value": "READ COMMITTED",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "READ COMMITTED",
    "database_value": "READ COMMITTED",
    "default_value": "READ COMMITTED",
    "description": "READ COMMITTED (Default) - Last epoch for reads and current epoch for writes.
SERIALIZABLE - Current epoch for reads and writes",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
    "parameter_name": "TransactionIsolationLevel",
    "source": "DEFAULT"
  }
]
```

PUT databases/:database_name/configuration

Sets one or more configuration parameters for the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Returns the parameter name, the requested value, and the result of the attempted change (Success or Failed).

Resource URL

```
https://<NODE>:5444/databases/:database_name/configuration
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

<i>user_id</i>	A database username.
<i>passwd</i>	The password for the username.
<i>parameter_name</i>	A parameter name and value combination for the parameter to be changed. Values must be URL encoded. You can include multiple name/value pairs to set multiple parameters with a single API call.

Example Request

PUT	<pre>https://<NODE>:5444/databases/testDB/configuration?user_id=dbadmin"&"passwd=vertica "&"JavaBinaryForUDx=%2Fusr%2Fbin%2Fjava"&"TransactionIsolationLevel=SERIALIZABLE</pre>
------------	---

Response:

```
[
  {
    "key": "JavaBinaryForUDx",
    "result": "Success",
    "value": "/usr/bin/java"
```

```
  },  
  {  
    "key": "TransactionIsolationLevel",  
    "result": "Success",  
    "value": "SERIALIZABLE"  
  }  
]
```

GET databases/:database_name/hosts

Returns the hostname/IP address, node name, and UP/DOWN status of each host associated with the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/databases/:database_name/hosts
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/databases/VMart/hosts</pre>
------------	--

Response:

```
{
  "body": [
    {
      "hostname": "10.20.100.247",
      "nodename": "v_vmart_node0001",
      "status": "UP",
      "ts": "2014-07-18T13:12:31.904191"
    },
    {
      "hostname": "10.20.100.248",
      "nodename": "v_vmart_node0002",
      "status": "UP",
      "ts": "2014-07-18T13:12:31.904209"
    },
    {
      "hostname": "10.20.100.249",
      "nodename": "v_vmart_node0003",
      "status": "UP",

```

```
    "ts": "2014-07-18T13:12:31.904215"  
  }  
],  
"href": "/databases/VMart/hosts",  
"links": [],  
"mime-type": "application/vertica.hosts.json-v2"  
}
```

POST databases/:database_name/hosts

Creates a job to add a host to the database identified by :database_name. This host must already be part of the cluster. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444//:database_name/hosts
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

user_id	A database username.
passwd	The password for the username.
hostname	The hostname to add to the database. This host must already be part of the cluster.

Example Request

POST	<pre>https://<NODE>:5444/databases/testDB/hosts?hostname=192.168.232.181&user_id=dbadmin"&"passwd=vertica</pre>
-------------	---

Response:

```
{
  "id": "AddHostToDatabase-testDB-2014-07-20 12:24:04.088812",
  "url": "/jobs/AddHostToDatabase-testDB-2014-07-20 12:24:04.088812"
}
```

DELETE databases/:database_name/hosts/:host_id

Creates a job to remove the host identified by `:host_id` from the database identified by `:database_name`. The `:database_name` is the value of the `name` field that the [GETdatabases](#) command returns. The `:host_id` is the value of the `host` field returned by [GETdatabases/:database_name](#).

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name/hosts/:host_id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

user_id	A database username.
passwd	A password for the username.

Example Request

DELETE	<pre>https://<NODE>:5444/databases/testDB/hosts/192.168.232.181?user_id=dbadmin"&"passwd=vertica</pre>
---------------	--

Response:

```
{
  "id": "RemoveHostFromDatabase-testDB-2014-07-20 13:41:15.646235",
  "url": "/jobs/RemoveHostFromDatabase-testDB-2014-07-20 13:41:15.646235"
}
```

POST databases/:database_name/hosts/:host_id/process

Creates a job to start the vertica process for the database identified by :database_name on the host identified by :host_id. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns. The :host_id is the value of the *host* field returned by [GETdatabases/:database_name](#).

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/:database_name/hosts/:host_id/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

POST	<pre>https://<NODE>:5444/databases/testDB/hosts/192.168.232.181/process</pre>
-------------	---

Response:

```
{
  "id": "StartDatabase-testDB-2014-07-20 13:14:03.968340",
  "url": "/jobs/StartDatabase-testDB-2014-07-20 13:14:03.968340"
}
```

GET databases/:database_name/license

Returns details about the database license being used by the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/:database_name/license
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

user_id	A database username.
passwd	The password for the username.

Example Request

GET	https://<NODE>:5444/VMart/license
-----	-----------------------------------

Response:

```
{
  "body": {
    "details": {
      "assigned_to": "Vertica Systems, Inc.",
      "grace_period": 0,
      "is_ce": false,
      "is_unlimited": false,
      "name": "vertica",
      "not_after": "Perpetual",
      "not_before": "2007-08-03"
    },
    "last_audit": {
      "audit_date": "2014-07-18 13:49:22.530105-04",
      "database_size_bytes": "814060522",

```

```
        "license_size_bytes": "536870912000",  
        "usage_percent": "0.00151630588248372"  
    }  
},  
"href": "/databases/VMart/license",  
"links": [],  
"mime-type": "application/vertica.license.json-v2"  
}
```

GET databases/:database_name/licenses

Returns details about all license being used by the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/:database_name/licenses
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

user_id	A database username.
passwd	The password for the username.

Example Request

GET	<pre>https://<NODE>:5444/VMart/licenses</pre>
------------	---

Response:

```
{
  "body": [
    {
      "details": {
        "assigned_to": "Vertica Systems, Inc.",
        "audit_date": "2014-07-19 21:35:25.111312",
        "is_ce": "False",
        "name": "vertica",
        "node_restriction": "",
        "not_after": "Perpetual",
        "not_before": "2007-08-03",
        "size": "500GB"
      },
      "last_audit": {
        "audit_date": "2014-07-19 21:35:26.318378-04",
```

```
        "database_size_bytes": "819066288",
        "license_size_bytes": "536870912000",
        "usage_percent": "0.00152562984824181"
    }
},
{
    "details": {
        "assigned_to": "Vertica Systems, Inc., FlexTable",
        "audit_date": "2014-07-19 21:35:25.111312",
        "is_ce": "False",
        "name": "com.vertica.flextable",
        "node_restriction": "",
        "not_after": "Perpetual",
        "not_before": "2007-08-03",
        "size": "500GB"
    },
    "last_audit": {
        "audit_date": "2014-07-19 21:35:25.111312",
        "database_size_bytes": 0,
        "license_size_bytes": 536870912000,
        "usage_percent": 0
    }
}
],
"href": "/databases/VMart/licenses",
"links": [],
"mime-type": "application/vertica.features.json-v2"
}
```

DELETE databases/:database_name/hosts/:host_id/process

Creates a job to stop the vertica process for the database identified by :database_name on the host identified by :host_id. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns. The :host_id is the value of the *host* field returned by [GETdatabases/:database_name](#).

Returns a job ID that can be used to determine the status of the job. See [GET jobs](#).

Note: If stopping the database on the hosts causes the database to no longer be k-safe, then the all database nodes may shut down.

Resource URL

```
https://<NODE>:5444/databases/:database_name/hosts/:host_id/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

DELETE	<pre>https://<NODE>:5444/databases/testDB/hosts/192.168.232.181/process</pre>
---------------	---

Response:

```
{
  "id": "StopDatabase-testDB-2014-07-20 13:02:08.453547",
  "url": "/jobs/StopDatabase-testDB-2014-07-20 13:02:08.453547"
}
```

POST databases/:database_name/hosts/:host_id/replace_with/:host_id_new

Creates a job to replace the host identified by `hosts/:host_id` with the host identified by `replace_with/:host_id`. Vertica performs these operations for the database identified by `:database_name`. The `:database_name` is the value of the *name* field that the [GETdatabases](#) command returns. The `:host_id` is the value of the *host* field as returned by [GET databases/:database_name](#). You can find valid replacement hosts using [GET hosts](#). The replacement host cannot already be part of the database. You must stop the vertica process on the host being replaced.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name/hosts/:host_id/replace_with/:host_id_new
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

user_id	A database username.
passwd	A password for the username.

Example Request

POST	<pre>https://<NODE>:5444/databases/testDB/hosts/192.168.232.180/replace_with/192.168.232.181?user_id=dbadmin"&"passwd=vertica</pre>
-------------	---

Response:

```
{
  "id": "ReplaceNode-testDB-2014-07-20 13:50:28.423509",
  "url": "/jobs/ReplaceNode-testDB-2014-07-20 13:50:28.423509"
```

```
}
```

GET databases/:database_name/nodes

Returns a comma-separated list of node IDs for the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/:database_name/nodes
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/VMart/nodes</pre>
------------	--

Response:

```
[
  {
    "database_id": "VMart",
    "node_id": "v_vmart_node0001,v_vmart_node0002,v_vmart_node0003",
    "status": "Unknown"
  }
]
```

GET databases/:database_name/nodes/:node_id

Returns details about the node identified by :node_id. The :node_id is one of the node IDs returned by [GET databases/:database_name/nodes](#).

Resource URL

```
https://<NODE>:5444/:database_name/nodes/:node_id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/databases/VMart/nodes/v_vmart_node0001</pre>
------------	---

Response:

```
{
  "db": "VMart",
  "host": "10.20.100.247",
  "name": "v_vmart_node0001",
  "state": "UP"
}
```

POST databases/:database_name/process

Creates a job to start the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Returns a job ID that can be used to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

epoch	Start the database from this epoch.
include	Include only these hosts when starting the database. Use a comma-separated list of hostnames.

Example Request

POST	<pre>https://<NODE>:5444/databases/:testDB/process</pre>
-------------	--

An example of the full request using cURL:

```
curl -X POST -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/:testDB/process
```

Response:

```
{
  "id": "StartDatabase-testDB-2014-07-20 12:41:46.061408",
  "url": "/jobs/StartDatabase-testDB-2014-07-20 12:41:46.061408"
}
```

GET databases/:database_name/process

Returns a state of UP or DOWN for the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Resource URL

```
https://<NODE>:5444/databases/:database_name/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/databases/VMart/process</pre>
------------	--

Response:

```
{  
  "state": "UP"  
}
```

DELETE databases/:database_name/process

Creates a job to stop the database identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

user_id	A database username.
passwd	The password for the username.

Example Request

DELETE	<pre>https://<NODE>:5444/databases/testDB/process?user_id=dbadmin"&"passwd=vertica</pre>
---------------	--

An example of the full request using cURL:

```
curl -X DELETE -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/:testDB/process?user_id=dbadmin"&"passwd=vertica
```

Response:

```
{
  "id": "StopDatabase-testDB-2014-07-20 12:46:04.406637",
  "url": "/jobs/StopDatabase-testDB-2014-07-20 12:46:04.406637"
}
```

POST databases/:database_name/rebalance/process

Creates a job to run a rebalance on the database identified by host identified by : database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name/rebalance/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

user_id	A database username.
passwd	A password for the username.

Example Request

POST	<pre>https://<NODE>:5444/databases/testDB/rebalance/process</pre>
-------------	---

Response:

```
{
  "id": "RebalanceData-testDB-2014-07-20 21:42:45.731038",
  "url": "/jobs/RebalanceData-testDB-2014-07-20 21:42:45.731038"
}
```

POST databases/:database_name/wla/process

Creates a job to run Workload Analyzer on the database identified by host identified by :database_name. The :database_name is the value of the *name* field that the [GETdatabases](#) command returns.

Returns a job ID that you can use to determine the status of the job. See [GET jobs](#).

Resource URL

```
https://<NODE>:5444/databases/:database_name/wla/process
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

user_id	A database username.
passwd	A password for the username.

Example Request

POST	<pre>https://<NODE>:5444/databases/testDB/wla/process</pre>
-------------	---

Response:

```
{
  "id": "AnalyzeWorkLoad-testDB-2014-07-20 21:48:27.972989",
  "url": "/jobs/AnalyzeWorkLoad-testDB-2014-07-20 21:48:27.972989"
}
```

Hosts

You can use these API calls to get information on the hosts in your cluster.

GET hosts	Returns a list of hosts in this cluster.
GET hosts/:hostid	Returns details for a specific host in this cluster.

GET hosts

Returns a list of the hosts in the cluster and the hardware, software, and network details about each host.

Resource URL

```
https://<NODE>:5444/hosts
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/hosts</pre>
------------	--

Response:

```
{
  "body": [
    {
      "cpu_info": {
        "cpu_type": " Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
        "number_of_cpus": 2
      },
      "host_id": "10.20.100.247",
      "hostname": "v_vmart_node0001.example.com",
      "max_user_proc": "3833",
      "nics": [
        {
          "broadcast": "10.20.100.255",
          "ipaddr": "10.20.100.247",
          "name": "eth0",
          "netmask": "255.255.255.0",
          "speed": "unknown"
        },
        {

```

```
        "broadcast": "255.255.255.255",
        "ipaddr": "127.0.0.1",
        "name": "lo",
        "netmask": "255.0.0.0",
        "speed": "locallink"
    }
],
"total_memory": 3833,
"vertica": {
    "arch": "x86_64",
    "brand": "vertica",
    "release": "20140716",
    "version": "8.1.0"
}
},
{
    "cpu_info": {
        "cpu_type": " Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
        "number_of_cpus": 2
    },
    "host_id": "10.20.100.248",
    "hostname": "v_vmart_node0002.example.com",
    "max_user_proc": "3833",
    "nics": [
        {
            "broadcast": "10.20.100.255",
            "ipaddr": "10.20.100.248",
            "name": "eth0",
            "netmask": "255.255.255.0",
            "speed": "unknown"
        },
        {
            "broadcast": "255.255.255.255",
            "ipaddr": "127.0.0.1",
            "name": "lo",
            "netmask": "255.0.0.0",
            "speed": "locallink"
        }
    ],
    "total_memory": 3833,
    "vertica": {
        "arch": "x86_64",
        "brand": "vertica",
        "release": "20140716",
        "version": "8.1.0"
    }
},
{
    "cpu_info": {
        "cpu_type": " Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
        "number_of_cpus": 2
    },
    "host_id": "10.20.100.249",
    "hostname": "v_vmart_node0003.example.com",
    "max_user_proc": "3833",
    "nics": [
        {
            "broadcast": "10.20.100.255",
            "ipaddr": "10.20.100.249",
            "name": "eth0",
```

```
        "netmask": "255.255.255.0",
        "speed": "unknown"
      },
      {
        "broadcast": "255.255.255.255",
        "ipaddr": "127.0.0.1",
        "name": "lo",
        "netmask": "255.0.0.0",
        "speed": "locallink"
      }
    ],
    "total_memory": 3833,
    "vertica": {
      "arch": "x86_64",
      "brand": "vertica",
      "release": "20140716",
      "version": "8.1.0"
    }
  }
},
"href": "/hosts",
"links": [
  "[:hostid]"
],
"mime-type": "application/vertica.hosts.json-v2"
}
```

GET hosts/:hostid

Returns hardware, software, and network details about the host identified by `:host_id`. You can find `:host_id` for each host using [GET hosts](#).

Resource URL

```
https://<NODE>:5444/hosts/:hostid
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/hosts/10.20.100.247</pre>
------------	--

Response:

```
{
  "body": {
    "cpu_info": {
      "cpu_type": " Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
      "number_of_cpus": 2
    },
    "hostname": "v_vmart_node0001.example.com",
    "max_user_proc": "3833",
    "nics": [
      {
        "broadcast": "10.20.100.255",
        "ipaddr": "10.20.100.247",
        "name": "eth0",
        "netmask": "255.255.255.0",
        "speed": "unknown"
      },
      {
        "broadcast": "255.255.255.255",
        "ipaddr": "127.0.0.1",

```

```
        "name": "lo",
        "netmask": "255.0.0.0",
        "speed": "loallink"
      }
    ],
    "total_memory": 3833,
    "vertica": {
      "arch": "x86_64",
      "brand": "vertica",
      "release": "20140716",
      "version": "8.1.0"
    }
  },
  "href": "/hosts/10.20.100.247",
  "links": [],
  "mime-type": "application/vertica.host.json-v2"
}
```

Jobs

You can use these API calls to get information on your database's jobs.

GET jobs	Returns a list of jobs the agent is tracking, along with their current status and exit codes.
GET jobs/:id	Returns the details (the saved output) for a specific job.

GET jobs

Returns a list of jobs being tracked by the agent and job details.

Jobs always start immediately. The `is_running` field is a Boolean value. If `is_running` is false, then the job is complete.

The `exit_code` details the status of the job. The `exit_code` is different for certain types of jobs:

- For Backup jobs:
 - 0 indicates success.
 - Any other number indicates a failure.
- For all other jobs:
 - -9 indicates success.
 - Any other number indicates a failure.

You can see details about failures in `/opt/vertica/log/agentStdMsg.log`.

Resource URL

```
https://<NODE>:5444/jobs
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/jobs</pre>
------------	---

Response:

```
{
  "body": [
    {
      "exit_code": 0,
      "id": "CreateBackup-VMart-1405012447.75",
      "is_running": false,
      "status": "unused",
      "ts": "1405012461.18"
    },
    {
      "exit_code": 1,
      "id": "CreateBackup-VMart-1405012454.88",
      "is_running": false,
      "status": "unused",
      "ts": "1405012455.18"
    }
  ],
  "href": "/jobs",
  "links": [
    "/:jobid"
  ],
  "mime-type": "application/vertica.jobs.json-v2"
}
```

GET jobs/:id

Gets the details for a specific job with the provided `:id`. You can determine the list of job `ids` using [GET jobs](#).

Details for a specific job are the same as the details provided for all jobs by [GET jobs](#).

Note: You must URL encode the `:id` as some IDs may contain spaces or other special characters.

Resource URL

```
https://<NODE>:5444/jobs/:id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/jobs/CreateBackup-VMart-1405012454.88</pre>
-----	--

Licenses

You can use these API calls to manage licenses for your database.

POST licenses	Uploads and applies a new license to this cluster.
GET licenses	Returns the license field that databases created on this cluster use.

POST licenses

Uploads and applies a license file to this cluster.

You must provide the license file as an HTTP POST form upload, identified by the name *license*. For example, you can use cURL:

```
curl -k --request POST -H "VerticaApiKey:ValidAPIKey" \  
https://v_vmart_node0001:5444/licenses --form "license=@vlicense.dat"
```

Resource URL

https://<NODE>:5444/licenses

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **admin** level security.

Parameters

None.

Example Request

POST	https://<NODE>:5444/licenses
-------------	------------------------------

Response:

There is no HTTP body response for successful uploads. A successful upload returns an HTTP 200/OK header.

GET licenses

Returns any license files that are used by this cluster when creating databases. License files must reside in `/opt/vertica/config/share`.

Resource URL

```
https://<NODE>:5444/licenses
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/licenses</pre>
------------	---

Response:

```
{
  "body": [
    {
      "comment": "Vertica license is valid",
      "end": "Perpetual",
      "grace": "0",
      "size": "1TB CE Nodes 3",
      "start": "2011-11-22",
      "status": true,
      "vendor": "Vertica Community Edition"
    }
  ],
  "href": "/license",
  "links": [],
  "mime-type": "application/vertica.license.json-v2"
}
```

Nodes

You can use these API calls to retrieve information on the nodes in your cluster.

GET nodes	Returns a list of nodes in this cluster.
GET nodes/:nodeid	Returns details for a specific node in this cluster.

GET nodes

Returns a list of nodes associated with this cluster.

Resource URL

```
https://<NODE>:5444/nodes
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/nodes</pre>
------------	--

Response:

```
{
  "body": [
    "node0001",
    "node0002",
    "node0003",
    "v_testdb_node0001",
    "v_testdb_node0002",
    "v_testdb_node0003",
    "v_vmart_node0001",
    "v_vmart_node0002",
    "v_vmart_node0003"
  ],
  "href": "/nodes",
  "links": [
    "/:nodeid"
  ],
  "mime-type": "application/vertica.nodes.json-v2"
}
```

GET nodes/:nodeid

Returns details about the node identified by :node_id. You can find the :node_id for each node using [GET nodes](#).

In the body field, the following information is detailed in comma-separated format:

- Node Name
- Host Address
- Catalog Directory
- Data Directory

Resource URL

```
https://<NODE>:5444/nodes/:node_id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/nodes/v_vmart_node0001</pre>
------------	---

Response:

```
{
  "body": [
    "v_vmart_node0001",
    "10.20.100.247,/home/dbadmin,/home/dbadmin"
  ],
  "href": "/nodes/v_vmart_node0001",
  "links": [],
}
```

```
}  "mime-type": "application/vertica.node.json-v2"
```

Webhooks

You can use these API calls to obtain information on, create, or delete webhooks.

GET webhooks	Returns a list of active webhooks.
POST webhooks/subscribe	Creates a new webhook.
DELETE webhooks/:subscriber_id	Deletes an existing webhook.

GET webhooks

Returns a list of active webhooks for this cluster.

Resource URL

```
https://<NODE>:5444/webhooks
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

GET	<pre>https://<NODE>:5444/webhooks</pre>
------------	---

Response:

```
{
  "body": [
    {
      "host": "192.168.232.1",
      "id": "79c1c8a18be02804b3d2f48ea6462909",
      "port": 80,
      "timestamp": "2014-07-20 22:54:09.829642",
      "url": "/gettest.htm"
    },
    {
      "host": "192.168.232.1",
      "id": "9c32cb0f3d2f9a7cb10835f1732fd4a7",
      "port": 80,
      "timestamp": "2014-07-20 22:54:09.829707",
      "url": "/getwebhook.php"
    }
  ],
  "href": "/webhooks",
  "links": [
    "/subscribe",
    "/:subscriber_id"
  ]
}
```

```
  ],  
  "mime-type": "application/vertica.webhooks.json-v2"  
}
```

POST webhooks/subscribe

Creates a subscription for a webhook.

Resource URL

```
https://<NODE>:5444/webhooks/subscribe
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

url	A URL to an application that accepts JSON messages from this cluster.
-----	---

Example Request

POST	<pre>https://<NODE>:5444/webhooks/subscribe?url=http%3A%2F%2Fexample.com%2Fgetwebhook.php</pre>
-------------	---

Response:

The response is not JSON encoded. The only text response is the ID of the webhook subscription. Additionally, an HTTP 200/OK header indicates success.

```
79c1c8a18be02804b3d2f48ea6462909
```

DELETE webhooks/:subscriber_id

Deletes the webhook identified by `:subscriber_id`. The `:subscriber_id` is the value of the `id` field that the [GET webhooks](#) command returns.

Resource URL

```
https://<NODE>:5444/webhooks/:subscriber_id
```

Authentication

Requires a [VerticaAPIKey](#) in the request header.

The API key must have **restricted** level security or higher.

Parameters

None.

Example Request

DELETE	<pre>https://<NODE>:5444/webhooks/79c1c8a18be02804b3d2f48ea6462909</pre>
---------------	--

Response:

There is no HTTP body response for successful deletes. A successful delete returns an HTTP 200/OK header.

Rest APIs for the Management Console

These API calls interact with Management Console nodes.

Alerts

GET alerts	Returns alerts for the current user.
----------------------------	--------------------------------------

Time Information

GET <code>mcTimeInfo</code>	Returns the current time for the MC server and the timezone of the location where the MC server is located.
---------------------------------------	---

MC-User-ApiKey

The MC-User-ApiKey is a user-specific key used with Management Console. Users must have an MC-User-ApiKey to interact with MC using the Rest API. All users with roles other than None automatically receive an MC-User-ApiKey.

This key grants users the same rights through the API that they have available through their MC roles. To interact with the MC, users pass the key in the request header for the API.

View the MC-User-ApiKey

If you are the database administrator, you can view the MC-User-ApiKey for all users. Individual users can view their own keys.

1. Connect to MC and go to MC Settings > User Management.
2. Select the user to view and click Edit. The user's key appears in the User API Key field.

GET alerts

Returns a list of MC alerts, their current status, and database properties.

Resource URL

```
https://<MC_NODE>:5450/webui/api/alerts
```

Authentication

Requires an [MC-User-Apikey](#) in the request header.

Filter Parameters

types	<p>The type of alert to retrieve. Valid values are:</p> <ul style="list-style-type: none">• info• notice• warning• error• critical• alert• emergency
category	<p>For information, see Thresholds Category Filter.</p>
db_name	<p>For information, see Database Name Category Filter.</p>
limit	<p>The maximum number of alerts to retrieve. If the limit is lower than the number of existing alerts, Vertica retrieves the most recent alerts. Used with the type parameter, Vertica retrieves up to the limit for each type. For example, for a limit of five and types of critical and emergency, you could receive up to ten total alerts.</p>
time_from	<p>The timestamp start point from which to retrieve alerts. You can use this parameter in combination with the time_to parameter to retrieve alerts for a specific time range. Values must be passed in the following format: <code>yyyy-MM-ddTHH:mm</code>.</p> <p>If you provide only the time_from parameter, and omit the time_to parameter, the response contains all alerts generated from the time_from parameter to the current time.</p>
time_to	<p>The timestamp end point from which to retrieve alerts. You can use this</p>

parameter in combination with the `time_from` parameter to retrieve alerts for a specific time range. Values must be passed in the following format: `yyyy-MM-ddTHH:mm`.

If you provide only the `time_to` parameter, and omit the `time_from` parameter, the response contains all alerts generated from the earliest possible time to the time passed in `time_to`.

Example Request

GET

```
https://<MC_NODE>:5450/webui/api/alerts
```

Request Alerts Using cURL

This example shows how you can request alerts using cURL. In this example, the `limit` parameter is set to '2' and the `types` parameters is set to `info` and `notice`:

```
curl -H "MC-User-APIKey: ValidUserKey" https://<MC_NODE>:5450/webui/api/alerts?limit=2&types=info,notice
```

Response:

```
[
  {
    "alerts": [
      {
        "id": 5502,
        "markedRead": false,
        "eventTypeCode": 0,
        "create_time": "2016-02-02 05:12:10.0",
        "updated_time": "2016-02-02 15:50:20.511",
        "severity": "warning",
        "status": 1,
        "nodeName": "v_vmart_node0001",
        "databaseName": "VMart",
        "databaseId": 1,
        "clusterName": "1449695416208_cluster",
        "description": "Warning: Low disk space detected (73% in use)",
        "summary": "Low Disk Space",
        "internal": false,
        "count": 3830
      },
      {
        "id": 5501,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2016-02-02 05:12:02.31",
        "updated_time": "2016-02-02 05:12:02.31",
      }
    ]
  }
]
```

```
    "severity": "notice",
    "status": 1,
    "databaseName": "VMart",
    "databaseId": 1,
    "clusterName": "1449695416208_cluster",
    "description": "Analyze Workload operation started on Database",
    "summary": "Analyze Workload operation started on Database",
    "internal": false,
    "count": 1
  }
],
"total_alerts": 190,
"request_query": "limit=2",
"request_time": "2016-02-02 15:50:26 -0500"
}
]
```

Request Alerts Within a Time Range

These examples show various ways in which you can request the same alert as in the preceding example, but within specified time ranges.

Request the alert within a specific time range, using the `time_from` and `time_to` parameters:

```
curl -H "MC-User-ApiKey: ValidUserKey" https://<MC_
NODE>:5450/webui/api/alerts?types=info,notice&time_from=2016-01-01T12:12&time_to=2016-02-01T12:12
```

Request the alert from a specific start time to the present using the `time_from` parameter:

```
curl -H "MC-User-ApiKey: ValidUserKey" https://<MC_
NODE>:5450/webui/api/alerts?types=info,notice&time_from=2016-01-01T12:12
```

Request the alert to a specific end point using the `time_to` parameter. When you use the `time_to` parameter without the `time_from` parameter, the `time_from` parameter defaults to the oldest alerts your MC contains:

```
curl -H "MC-User-ApiKey: ValidUserKey" https://<MC_
NODE>:5450/webui/api/alerts?types=info,notice&time_to=2016-01-01T12:12
```

GET mcTimeInfo

Returns the current time for the MC server and the timezone where the MC server is located.

Resource URL

```
https://<MC_NODE>:5450/webui/api/mcTimeInfo
```

Authentication

Requires an [MC-User-Apikey](#) in the request header.

Parameters

None.

Example Request

GET	<code>https://<MC_NODE>:5450/webui/api/mcTimeInfo</code>
-----	--

This example shows how you can request MC time information using cURL:

```
curl -H "MC-User-ApiKey: ValidUserKey" https://<MC_NODE>:5450/webui/api/mcTimeInfo
```

Response:

```
{"mc_current_time":"Tue, 2000-01-01 01:02:03 -0500","mc_timezone":"US/Eastern"}
```

Thresholds Category Filter

Returns a list of alerts related to threshold settings in MC.

Resource URL

```
https://<MC_NODE>:5450/webui/api/alerts?category=thresholds
```

Authentication

Requires an [MC-User-Apikey](#) in the request header.

Example Request

GET

```
https://<MC_NODE>:5450/webui/api/alerts?category=thresholds
```

This example shows how you can request alerts on thresholds using cURL:

```
curl -H "MC-User-ApiKey: ValidUserKey" https://<MC_NODE>:5450/webui/api/alerts?category=thresholds
```

Response:

```
[
  {
    "alerts": [
      {
        "id": 33,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-10 10:28:41.332",
        "updated_time": "2015-11-10 10:28:41.332",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": " Database: mydb Lower than threshold Node Disk I/O 10 % v_mydb_node0002
;1.6% v_mydb_node0002 ;1.4% v_mydb_node0002 ;2.3% v_mydb_node0002 ;1.13% v_mydb_node0002 ;1.39%
v_mydb_node0001 ;3.78% v_mydb_node0003 ;1.79% ",
        "summary": "Threshold : Node Disk I/O < 10 %",
        "internal": false,
        "count": 1
      },
      {
        "id": 32,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-10 10:28:40.975",
        "updated_time": "2015-11-10 10:28:40.975",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": " Database: mydb Lower than threshold Node Memory 10 % v_mydb_node0002
;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002
;5.48% v_mydb_node0003 ;4.53% ",
        "summary": "Threshold : Node Memory < 10 %",
        "internal": false,
        "count": 1
      },
      {
        "id": 31,
        "markedRead": false,
        "eventTypeCode": 2,
```

```
    "create_time":"2015-11-10 10:28:40.044",
    "updated_time":"2015-11-10 10:28:40.044",
    "severity":"warning",
    "status":1,
    "databaseName":"mydb",
    "databaseId":1,
    "clusterName":"1446668057043_cluster",
    "description":" Database: mydb Lower than threshold Node CPU 10 %   v_mydb_node0002 ;1.4%
v_mydb_node0002 ;1.64%   v_mydb_node0002 ;1.45%   v_mydb_node0002 ;2.49%   ",
    "summary":"Threshold : Node CPU < 10 %",
    "internal":false,
    "count":1
  },
  {
    "id":30,
    "markedRead":false,
    "eventTypeCode":2,
    "create_time":"2015-11-10 10:28:34.562",
    "updated_time":"2015-11-10 10:28:34.562",
    "severity":"warning",
    "status":1,
    "databaseName":"mydb",
    "databaseId":1,
    "clusterName":"1446668057043_cluster",
    "description":" Database: mydb Exceed threshold Node Disk Usage 60 %   v_mydb_node0001
;86.41%   ",
    "summary":"Threshold : Node Disk Usage > 60 %",
    "internal":false,
    "count":1
  }
],
"total_alerts":4,
"request_query":"category=thresholds",
"request_time":"2015-11-10 10:29:17.129"
}
]
```

See Also

- [Combining Sub-Category Filters with Category Filters](#)

Database Name Category Filter

Returns a list of MC alerts for a specific database.

Resource URL

```
https://<MC_NODE>:5450/webui/api/alerts?db_name=<database_name>
```

Authentication

Requires an [MC-User-Apikey](#) in the request header.

Example Request

GET	<code>https://<MC_NODE>:5450/webui/api/alerts?db_name="mydb"</code>
------------	---

This example shows how you can view alerts on a specific database using cURL:

```
curl -H "MC-User-ApiKey: ValidUserKey" https://<MC_NODE>:5450/webui/api/alerts?db_name="mydb"
```

Response:

```
[
  {
    "alerts": [
      {
        "id": 9,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 15:10:53.391",
        "updated_time": "2015-11-05 15:10:53.391",
        "severity": "notice",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": "Workload analyzed successfully",
        "summary": "Analyze Workload operation has succeeded on Database",
        "internal": false,
        "count": 1
      },
      {
        "id": 8,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 15:10:31.16",
        "updated_time": "2015-11-05 15:10:31.16",
        "severity": "notice",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": "Analyze Workload operation started on Database",
        "summary": "Analyze Workload operation started on Database",
        "internal": false,
        "count": 1
      },
      {

```

```
    "id":7,
    "markedRead":false,
    "eventTypeCode":2,
    "create_time":"2015-11-05 00:15:00.204",
    "updated_time":"2015-11-05 00:15:00.204",
    "severity":"alert",
    "status":1,
    "databaseName":"mydb",
    "databaseId":1,
    "clusterName":"1446668057043_cluster",
    "description":"Workload analyzed successfully",
    "summary":"Analyze Workload operation has succeeded on Database",
    "internal":false,
    "count":1
  },
  {
    "id":6,
    "markedRead":false,
    "eventTypeCode":2,
    "create_time":"2015-11-04 15:14:59.344",
    "updated_time":"2015-11-04 15:14:59.344",
    "severity":"notice",
    "status":1,
    "databaseName":"mydb",
    "databaseId":1,
    "clusterName":"1446668057043_cluster",
    "description":"Workload analyzed successfully",
    "summary":"Analyze Workload operation has succeeded on Database",
    "internal":false,
    "count":1
  },
  {
    "id":5,
    "markedRead":false,
    "eventTypeCode":2,
    "create_time":"2015-11-04 15:14:38.925",
    "updated_time":"2015-11-04 15:14:38.925",
    "severity":"notice",
    "status":1,
    "databaseName":"mydb",
    "databaseId":1,
    "clusterName":"1446668057043_cluster",
    "description":"Analyze Workload operation started on Database",
    "summary":"Analyze Workload operation started on Database",
    "internal":false,
    "count":1
  },
  {
    "id":4,
    "markedRead":false,
    "eventTypeCode":0,
    "create_time":"2015-11-04 15:14:33.0",
    "updated_time":"2015-11-05 16:26:17.978",
    "severity":"notice",
    "status":1,
    "nodeName":"v_mydb_node0001",
    "databaseName":"lmydb",
    "databaseId":1,
    "clusterName":"1446668057043_cluster",
    "description":"Workload analyzed successfully",
```

```
    "summary": "Analyze Workload operation has succeeded on Database",
    "internal": false,
    "count": 1
  },
  {
    "id": 3,
    "markedRead": false,
    "eventTypeCode": 2,
    "create_time": "2015-11-04 15:14:32.806",
    "updated_time": "2015-11-04 15:14:32.806",
    "severity": "info",
    "status": 1,
    "hostIp": "10.20.100.64",
    "nodeName": "v_mydb_node0003",
    "databaseName": "mydb",
    "databaseId": 1,
    "clusterName": "1446668057043_cluster",
    "description": "Agent status is UP on IP 127.0.0.1",
    "summary": "Agent status is UP on IP 127.0.0.1",
    "internal": false,
    "count": 1
  },
  {
    "id": 2,
    "markedRead": false,
    "eventTypeCode": 2,
    "create_time": "2015-11-04 15:14:32.541",
    "updated_time": "2015-11-04 15:14:32.541",
    "severity": "info",
    "status": 1,
    "hostIp": "10.20.100.63",
    "nodeName": "v_mydb_node0002",
    "databaseName": "mydb",
    "databaseId": 1,
    "clusterName": "1446668057043_cluster",
    "description": "Agent status is UP on IP 127.0.0.1",
    "summary": "Agent status is UP on IP 127.0.0.1",
    "internal": false,
    "count": 1
  },
  {
    "id": 1,
    "markedRead": false,
    "eventTypeCode": 2,
    "create_time": "2015-11-04 15:14:32.364",
    "updated_time": "2015-11-04 15:14:32.364",
    "severity": "info",
    "status": 1,
    "hostIp": "10.20.100.62",
    "nodeName": "v_mydb_node0001",
    "databaseName": "mydb",
    "databaseId": 1,
    "clusterName": "1446668057043_cluster",
    "description": "Agent status is UP on IP 127.0.0.1",
    "summary": "Agent status is UP on IP 127.0.0.1",
    "internal": false,
    "count": 1
  }
],
"total_alerts": 9,
```

```
"request_query": "db_name=mydb",  
"request_time": "2015-11-05 16:26:21.679"  
}  
]
```

Combining Sub-Category Filters with Category Filters

You can combine category filters with sub-category filters, to obtain alert messages for specific thresholds you set in MC. You can also use sub-category filters to obtain information about alerts on specific resource pools in your database.

Sub-Category Filters

You can use the following sub-category filters with the category filters. Sub-category filters are case sensitive and must be lowercase.

Sub-Category Filter	Alerts Related to Threshold Value Set For:
threshold_node_cpu	Node CPU
threshold_node_memory	Node Memory
threshold_node_disk_usage	Node Disk Usage
threshold_node_diskio	Node Disk I/O
threshold_node_cpuio	Node CPU I/O Wait
threshold_node_rebootrate	Node Reboot Rate
threshold_netio	Network I/O Error
threshold_query_queued	Queued Query Number
threshold_query_failed	Failed Query Number
threshold_query_spilled	Spilled Query Number
threshold_query_retried	Retried Query Number
threshold_query_runtime	Query Running Time

Resource Pool-Specific Sub-Category Filters

To retrieve alerts for a specific resource pool, you can use sub-category filters in combination with the following category filters:

- `thresholds`
- `rp_name`

If you use these sub-category filters without the `rp_name` filter, the query retrieves alerts for all resource pools in your database.

Sub-Category Filter	Alerts Related to Threshold Value Set For:
<code>threshold_rp_query_max_time</code>	Queries reaching the maximum allowed execution time.
<code>threshold_rp_query_resource_reject</code>	The number of queries with resource rejections.
<code>threshold_rp_query_queue_time</code>	The number of queries that ended because of queue time exceeding a limit.
<code>threshold_rp_query_run_time</code>	The number of queries that ended because of run time exceeding a limit.
<code>threshold_rp_memory</code>	The minimum allowed resource pool size.
<code>threshold_rp_max_memory</code>	The maximum allowed resource pool size.

Authentication

Requires an [MC-User-Apikey](#) in the request header.

Example Request

GET	<pre>https://<MC_NODE>:5450/webui/api/alerts?category=thresholds&subcategory=<subcategory_filter></pre>
-----	---

Combine the Thresholds Category Filter with a Sub-Category Filter

This example shows how you can request alerts using cURL with the thresholds category filter and a sub-category filter. You apply the following filters:

- thresholds
- threshold_node_cpu

```
curl -H "MC-User-APIKey: ValidUserKey" https://<MC_
NODE>:5450/webui/api/alerts?category=thresholds&subcategory=threshold_node_cpu
```

Response:

```
[
  {
    "alerts": [
      {
        "id": 11749,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 11:04:43.997",
        "updated_time": "2015-11-05 11:04:43.997",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 105,
        "clusterName": "1443122180317_cluster",
        "description": " Database: mydb Lower than threshold Node CPU 10 %   v_mydb_node0002
;1.03% v_mydb_node0003 ;0.9% v_mydb_node0001 ;1.36%   ",
        "summary": "Threshold : Node CPU < 10 %",
        "internal": false,
        "count": 1
      },
      {
        "id": 11744,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 10:59:46.107",
        "updated_time": "2015-11-05 10:59:46.107",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb2",
        "databaseId": 106,
        "clusterName": "1443552354071_cluster",
        "description": " Database: mydb2 Lower than threshold Node CPU 10 %   v_mydb2_node0002
;0.83% v_mydb2_node0001 ;1.14%   ",
        "summary": "Threshold : Node CPU < 10 %",
        "internal": false,
        "count": 1
      }
    ]
  }
]
```

```
    ],  
    "total_alerts":2,  
    "request_query":"category=thresholds&subcategory=threshold_node_cpu",  
    "request_time":"2015-11-05 11:05:28.116"  
  }  
]
```

Request an Alert On a Specific Resource Pool

This example shows how you can request alerts using cURL on a specific resource pool. The name of the resource pool is resourcepool1. You apply the following filters:

- thresholds
- rp_name
- threshold_rp_query_run_time

```
curl -H "MC-User-APIKey: ValidUserKey" https://<MC_<br>NODE>:5450/webui/api/alerts?category=thresholds&subcategory=threshold_rp_query_run_time&rp_<br>name=resourcepool1
```

Response:

```
[  
  {  
    "alerts":[  
      {  
        "id":6525,  
        "markedRead":false,  
        "eventTypeCode":2,  
        "create_time":"2015-11-05 14:25:36.797",  
        "updated_time":"2015-11-05 14:25:36.797",  
        "severity":"warning",  
        "status":1,  
        "databaseName":"mydb",  
        "databaseId":106,  
        "clusterName":"1443552354071_cluster",  
        "description":" Resource Pool: resourcepool1 Threshold Name: Ended Query with Run Time<br>Exceeding Limit Time Interval: 14:20:36 to 14:25:36 Threshold Value: 0 min(s) Actual Value: 2186<br>query(s) ",  
        "summary":"Resource Pool: resourcepool1; Threshold : Ended Query with Run Time Exceeding<br>Limit > 0 min(s)",  
        "internal":false,  
        "count":1  
      },  
      {  
        "id":6517,  
        "markedRead":false,  
        "eventTypeCode":2,  
        "create_time":"2015-11-05 14:20:39.541",  
        "updated_time":"2015-11-05 14:20:39.541",
```

```
    "severity": "warning",
    "status": 1,
    "databaseName": "mydb",
    "databaseId": 106,
    "clusterName": "1443552354071_cluster",
    "description": " Resource Pool: resourcepool1 Threshold Name: Ended Query with Run Time
Exceeding Limit Time Interval: 14:15:39 to 14:20:39 Threshold Value: 0 min(s) Actual Value: 2259
query(s) ",
    "summary": "Resource Pool: resourcepool1; Threshold : Ended Query with Run Time Exceeding
Limit > 0 min(s)",
    "internal": false,
    "count": 1
  }
],
"total_alerts": 14,
"request_query": "category=thresholds&subcategory=threshold_rp_query_run_time&rp_
name=resourcepool1",
"request_time": "2015-11-05 11:07:43.988"
}
]
```


Using Vertica on the Cloud

Welcome to the Vertica on the Cloud guide. This section explains how you can create Vertica clusters on different cloud platforms.

This document assumes that you are familiar with the cloud environment on which you will create your Vertica cluster.

Note: The cloud versions of Vertica are released shortly after the Enterprise version. Please check the Vertica version listed on your cloud provider to confirm which version of Vertica on the Cloud you are using.

Vertica on Amazon Web Services

Welcome to the Vertica on Amazon Web Services (AWS) guide. This section explains how you can create and manage Vertica clusters on AWS.

In This Section

Overview of Vertica on Amazon Web Services (AWS)

Vertica clusters on AWS operate on Amazon Machine Instances (AMI). The instructions in this document apply to AMIs built with Vertica Version 8.1.x.

Key Topics

The following are links to key topics within the Vertica on AWS documentation:

- If you created a standard 3 node cluster using a cloud formation template and are looking for information about how to connect, visit [Connecting to an Instance](#).
- If you have a running cluster, and want to add nodes to it, visit [Adding Nodes to a Running AWS Cluster](#).
- If you are looking for information about backup and restore operations, visit [Backup and Restore Vertica on AWS](#).

Supported Instance Types

Vertica supports a range of Amazon Web Services (AWS) instance types, each optimized for different purposes. Choose the instance type that best matches your performance and price needs as a user:

Optimization	Type
General Purpose	m4.4xlarge m4.10xlarge
Compute	c3.4xlarge c3.8xlarge c4.4xlarge c4.8xlarge
Memory	r3.4xlarge r3.8xlarge r4.4xlarge r4.8xlarge r4.16xlarge
Storage (supports ephemeral storage)	i2.4xlarge i2.8xlarge
Dense-storage (supports ephemeral storage)	d2.4xlarge d2.8xlarge

Note: Data stored on the ephemeral drives of a storage-optimized instance exists only while that instance is powered on. After powering off a storage-optimized system, data on ephemeral drives is lost.

More Information

For more information about Amazon cluster instances and their limitations, see the [Amazon documentation](#).

Vertica AMI Operating Systems

Micro Focus International plc provides Vertica and Management Console (MC) AMIs in multiple operating systems.

Vertica AMIs are available in the following operating systems:

- Red Hat 7.3
- Centos 7.3

MC AMIs are available in the following operating systems:

- Red Hat 7.3
- Centos 7.3

Vertica AMI Sleep C-States

By default, the following instances have their processor C-states set to a value of 1 in the Vertica AMI:

- c4.8xlarge
- d2.8xlarge
- m4.10xlarge

This measure is meant to improve performance by limiting the sleep states an instance running Vertica will use.

More Information

For more information about sleep states, visit the [AWS Documentation](#).

EC2 Launch Options

You can create one or multi node Vertica clusters on EC2 instances. Search for Vertica in the [Amazon Marketplace](#) to view these launch options.

1-Click Launch

This option deploys a single-node Vertica AMI with default settings.

Manual Launch

This option deploys a single-node Vertica AMI or a 3-node Vertica cluster with more customizable options.

Service Catalog

This option deploys a single-node AMI or a multi-node cluster for an administrator to create and distribute to end users.

Supported Features

Vertica on AWS supports a number of Amazon features and comes with a number of pre installed Vertica packages.

Enhanced Networking

Vertica AMIs support the AWS enhanced networking feature. Micro Focus recommends you use Enhanced Networking for optimal performance.

For details, see [Enabling Enhanced Networking on Linux Instances in a VPC](#) in the AWS documentation .

Packages

Vertica AMIs come with the following Vertica packages pre-installed:

- [Vertica Place](#)
- [Vertica Pulse](#)

Command Line Interface

Vertica AMIs support Amazon Command Line Interface (CLI).

For information about the Amazon CLI, visit the [AWS Documentation](#).

Elastic Load Balancing

You can use elastic load balancing (ELB) for queries up to one hour. When enabling ELB, ensure that the timer is configured to 3600 seconds.

For information about ELB, refer to [Amazon documentation](#).

Configure Your Network

Before you create your cluster, you must configure the network on which Vertica will run. Vertica requires a number of specific network configurations to operate on AWS. You may also have specific network configuration needs beyond the default Vertica settings.

Configure the following Amazon EC2 features in preparation for instance creation:

- [Security Group](#)
- [Access Control List](#)
- [Key Pairs](#)
- [Placement Group](#)
- [Internet Gateway](#)
- [Elastic IP](#)
- [Virtual Private Cloud](#)

Creating a Placement Group

Create a placement group to ensure that your nodes will be properly co-located.

More Information

For information about what a placement group is, as well as how to create one, visit the [AWS documentation](#).

Creating a Key Pair

Use a key pair to perform the following:

- Authenticate your connection as dbadmin to your instances from outside your cluster.
- Install and configure Vertica on your AWS instances.

Create an AWS key pair.

More Information

For information about what a key pair is, as well as how to create one, visit the [AWS documentation](#).

Creating a Virtual Private Cloud

A Vertica cluster on AWS must be logically located in the same network. This is similar to placing the nodes of an on-premises cluster within the same network. Create a virtual private cloud (VPC) to ensure the nodes in your cluster will be able to communicate with each other within AWS.

Create a **Single Public Subnet** VPC with the following configurations:

- Assign a Network Access Control List (ACL) that is appropriate to your situation. [The default ACL](#) does not provide a high level of security.
- Enable DNS resolution and Enable DNS hostname support for instances launched in this VPC.
- Add the required [network inbound and outbound rules to the Network ACL associated to the VPC](#).

Note: A Vertica cluster must be operated within a single availability zone.

More Information

For information about a VPC, as well as how to create one, visit the [AWS documentation](#).

Network ACL Settings

Vertica requires the following basic network access control list (ACL) settings on an AWS instance running the Vertica AMI. Micro Focus recommends that you secure your network with additional ACL settings that are appropriate to your situation; the default ACL does not provide a high level of security.

Inbound Rules

Type	Protocol	Port Range	Use	Source	Allow/Deny
SSH	TCP (6)	22	SSH (Optional -- for access to your cluster from outside your VPC)	User Specific	Allow
Custom TCP Rule	TCP (6)	5450	MC (Optional -- for MC running outside of your VPC)	User Specific	Allow
Custom TCP Rule	TCP (6)	5433	SQL Clients (Optional -- for access to your cluster from SQL clients)	User Specific	Allow
Custom TCP Rule	TCP (6)	50000	Rsync (Optional -- for backup outside of your VPC)	User Specific	Allow
Custom TCP Rule	TCP (6)	1024-65535	Ephemeral Ports (Needed if you use any of the above)	User Specific	Allow
ALL Traffic	ALL	ALL	N/A	0.0.0.0/0	Deny

Outbound Rules

Type	Protocol	Port Range	Use	Source	Allow/Deny
Custom TCP Rule	TCP (6)	0 - 65535	Ephemeral Ports	0.0.0.0/0	Allow

You can use the entire port range specified in the table above, or find your specific ephemeral ports by entering the following command:

```
cat /proc/sys/net/ipv4/ip_local_port_range
```

More Information

For detailed information on network ACLs within AWS, refer to [Amazon's documentation](#).

For detailed information on ephemeral ports within AWS, refer to [Amazon's documentation](#).

Creating and Assigning an Internet Gateway

When you create a VPC, an internet gateway is automatically assigned to it. You can use that gateway, or you can assign your own. If you are using the default internet gateway, continue with the next procedure, [Creating a Security Group](#).

Otherwise, create an internet gateway specific to your needs. Associate your internet gateway with your VPC and subnet.

More Information

For information about what an internet gateway is, as well as how to create one, visit the [AWS documentation](#).

Assigning an Elastic IP

An elastic IP is an unchanging IP address that you can use to connect to your cluster externally. Micro Focus recommends you assign a single elastic IP to a node in your cluster. You can then connect to other nodes in your cluster from your primary node using their internal IP addresses dictated by your VPC settings.

Create an elastic IP address.

More Information

For information about what an elastic IP is, as well as how to create one, visit the [AWS documentation](#).

Creating a Security Group

The Vertica AMI has specific security group requirements. When you create a Virtual Private Cloud (VPC), AWS automatically creates a default security group and assigns it to the VPC. You can use the default security group, or you can name and assign your own.

Create and name your own security group with the following basic security group settings. You may make additional modifications based on your specific needs.

Inbound

Type	Use	Protocol	Port Range	Source	IP
SSH		TCP	22	My IP	Limited IP range 169.24.165.0/24 or 0.0.0.0/0 for all internet access.
DNS (UDP)		UDP	53	My IP	10.0.0.0/24
Custom UDP	Spread	UDP	4803 and 4804	My IP	10.0.0.0/24
Custom TCP	Spread	TCP	4803	My IP	10.0.0.0/24
Custom TCP	VSQL/SQL	TCP	5433	My IP	Limited IP range 169.24.165.0/24 or 0.0.0.0/0 for all internet access.
Custom TCP	Inter-node Communication	TCP	5434	My IP	10.0.0.0/24
Custom TCP		TCP	5444	My IP	10.0.0.0/24
Custom	MC	TCP	5450	My IP	Limited IP range

TCP					169.24.165.0/24 or 0.0.0.0/0 for all internet access.
Custom TCP		TCP	48073	My IP	10.0.0.0/24
Custom TCP	Rsync	TCP	50000	My IP	10.0.0.0/24
ICMP	Installer	Echo Reply	N/A	My IP	10.0.0.0/24
ICMP	Installer	Traceroute	N/A	My IP	10.0.0.0/24

All ports must have a rule to open in the subnetCDIR level to allow nodes to be interconnected. For example, 10.11.12.0/24.

Note: In Management Console (MC), the Java IANA discovery process uses port 7 once to detect if an IP address is reachable before the database import operation. Vertica tries port 7 first. If port 7 is blocked, Vertica switches to port 22.

Outbound

Type	Protocol	Port Range	Destination	IP
All TCP	TCP	0-65535	Anywhere	0.0.0.0/0
All ICMP	ICMP	0-65535	Anywhere	0.0.0.0/0
All UDP	UDP	0-65535	Anywhere	0.0.0.0/0

More Information

For information about what a security group is, as well as how to create one, visit the [AWS documentation](#).

Installing and Running Vertica on AWS

Once you have configured your network, you are ready to create your AWS instances and install vertica. Follow these procedures to install and run Vertica on AWS.

Related Topics

- [Configuring and Launching an Instance](#)
- [Connecting to an Instance](#)
- [Preparing Instances](#)
- [Configuring Storage](#)
- [Forming a Cluster](#)
- [After Your Cluster Is Up and Running](#)
- [Initial Installation and Configuration](#)
- [Using Management Console \(MC\) on AWS](#)

Configuring and Launching an Instance

Once you have Configured your network settings on AWS, you are ready to configure and launch the instances that you will later install vertica onto. An Elastic Compute Cloud (EC2) instance without a Vertica AMI is similar to a traditional host. Just like with an on-premises cluster, you must prepare and configure your cluster and network at the hardware level before you can install Vertica.

When you create an EC2 instance on AWS using a Vertica AMI, the instance includes the Vertica software and a standard recommended configuration. The Vertica AMI acts as a template, requiring fewer configuration steps. Vertica recommends that you use the Vertica AMI as is – without modification.

Using the network configurations you created earlier, configure an instance in AWS:

1. Select the Vertica AMI from the AWS marketplace.
2. Select a [supported instance type](#).
3. Specify the number of instances you want to launch. A Vertica cluster uses identically configured instances of the same type. You cannot mix instance types.
4. Choose your VPC.

Note: Not all data centers support VPC. If you receive an error message that states "VPC is not currently supported...", choose a different region and zone (for example, choose us-east-1e rather than us-east-1c).

5. Assign a [Placement Group](#).

Add Storage:

1. Add storage to your instances based on your needs. Consider the following:
 - Micro Focus International plc recommends that you add a number of drives equal to the number of physical cores in your instance. For example, for a c3.8xlarge, add eight drives. For an r3.4xlarge, add four drives.
 - Micro Focus International plc does not recommend that you store your information on the root volume.
 - For optimal performance with EBS volumes, [Amazon recommends that you configure them in a RAID 0 array](#) on each node in your cluster.
 - Micro Focus International plc recommends that you create your storage when you create your instances. This allows AWS to optimize storage performance.

Configure Security Group:

1. Choose between your previously configured or the default Security Group.
2. Choose the key value pairs you intend to use with Vertica.

Launch Instances:

1. Verify that your instances are running.

Connecting to an Instance

Using your private key, connect to your cluster through the instance you attached an elastic IP to.

1. As the dbadmin user, type the following command, substituting your ssh key:

```
# ssh --ssh-identity <ssh key> dbadmin@elasticipaddress
```

2. Select **Instances** from the Navigation panel.
3. Select the instance that is attached to the Elastic IP.
4. Click **Connect**.
5. On **Connect to Your Instance**, choose one of the following options:
 - **A Java SSH Client directly from my browser**—Add the path to your private key in the field **Private key path**, and click **Launch SSH Client**.
 - **Connect with a standalone SSH client**—Follow the steps required by your standalone SSH client.

Connecting to an Instance from Windows Using Putty

If you connect to the instance from the Windows operating system, and plan to use Putty:

1. Convert your key file using PuTTYgen.
2. Connect with Putty or WinSCP (connect via the elastic IP), using your converted key (i.e., the *.ppk file).
3. Move your key file (the *.pem file) to the root dir using Putty or WinSCP.

Preparing Instances

After you create your instances, you need to prepare them for cluster formation. Prepare your instances by adding your AWS .pem key and your Vertica license.

By default, each AMI includes a Community Edition license. Once Vertica is installed, you can find the license at this location:

```
/opt/vertica/config/licensing/vertica_community_edition.license.key
```

1. As the dbadmin user, copy your *pem file (from where you saved it locally) onto your primary instance.

Depending upon the procedure you use to copy the file, the permissions on the file may change. If permissions change, the `install_vertica` script fails with a message similar to the following:

```
FATAL (19): Failed Login Validation 10.0.3.158, cannot resolve or connect to host as root.
```

If you receive a failure message, enter the following command to correct permissions on your *pem file:

```
chmod 600 /<name-of-pem>.pem
```

2. Copy your Vertica license over to your primary instance, placing it in your home directory or other known location.

Configuring Storage

Micro Focus recommends that you do *not* store your information on the root volume, especially your data and catalog directories. Instead, use dedicated EBS volumes for node storage. To take advantage of bursting, limit EBS volumes to 1TB or less. When configuring your storage, make sure to use a [supported file system](#).

For best performance, you can combine multiple EBS volumes into RAID-0. Vertica provides a shell script, which automates the storage configuration process.

Determining Volume Names

Before you combine volumes for storage, make note of your volume names so that you can alter the `configure_aws_software_raid.sh` shell script. You can find your volumes with the following commands:

```
cd /dev  
ls
```

Your volumes start with `xvd`.

Important: Ignore your root volume. Do not include any of your root volumes in the RAID creation process.

Combining Volumes for Storage

Follow these sample steps to combine your EBS volumes into RAID 0 using the `configure_aws_software_raid.sh` shell script.

1. Edit the `/opt/vertica/sbin/configure_aws_software_raid.sh` shell file as follows:
 - a. Comment out the `safety exit` command at the beginning .
 - b. Change the sample volume names to your own volume names, which you noted previously. Add more volumes, if necessary.
2. Run the `/opt/vertica/sbin/configure_aws_software_raid.sh` shell file. Running this file creates a RAID 0 volume and mounts it to `/vertica/data`.
3. Change the owner of the newly created volume to `dbadmin` with `chown` .
4. Repeat steps 1-3 for each node on your cluster.

More Information

For more information about EBS storage, refer to the [Amazon documentation](#).

Forming a Cluster

Use the `install_vertica` script to combine two or more individual instances and create a cluster.

Check the **My Instances** page for a list of current instances and their associated IP addresses. You need these IP addresses when you run the `install_vertica` script.

Combining Instances

The following example combines instances using the `install_vertica` script.

While connected to your primary instance, enter the following command to combine your instances into a cluster. Substitute the IP addresses for your instances and include your root `*pem` file name.

```
sudo /opt/vertica/sbin/install_vertica --hosts  
10.0.11.164,10.0.11.165,10.0.11.166 --dba-user-password-disabled --  
point-to-point --data-dir /vertica/data --ssh-identity ~/<name-of-  
pem>.pem --license <license.file>
```

Note: If you are using Community Edition, which limits you to three instances, you can simply specify `-L CE` with no license file.

When you issue `install_vertica` or `update_vertica` on an AMI, make sure to use the `--point-to-point` parameter. `--point-to-point` is the default when using AWS, so even if you don't specify `--point-to-point`, the installation is in effect. This parameter configures spread to use direct point-to-point communication between all Vertica nodes, which is a requirement for clusters on AWS. If you do not use the parameter, you receive an error telling you that you must use point-to-point communication on AWS.

Once you have combined your instances, Micro Focus International plc recommends deleting your `*pem` key from your cluster with the `shred` command to reduce security risks:

```
shred examplekey.pem
```

Important: You will need your `.pem` key to perform future Vertica updates.

Once you have formed a cluster, [Create a database](#).

Considerations When Using the `install_vertica` or `update_vertica` Scripts

- By default, the installer assumes that you have mounted your storage to `/vertica/data`. To specify another location, use the `--data-dir` argument. Micro Focus does not recommend that you store your data on the root drive.
- Password logons present a security risk on AWS. Include the parameter `--dba-user-password-disabled` so that the installer does not prompt for a password for the database user.

For complete information on the `install_vertica` script and its parameters, see the Installation Guide, specifically the section, *About the `install_vertica` Script*.

After Your Cluster Is Up and Running

Once your cluster is configured and running:

1. [Create a database](#). When Vertica was installed for you, a Vertica database administrator was created, dbadmin. You can use this pre-created dbadmin user to create and start a database. Refer to the Vertica Installation Guide for information on the dbadmin administrator.
2. Configure a database. Refer to the Vertica Administrator's Guide for information on configuring a database.
3. Refer to the full documentation set for Vertica for other tasks.

Safe Shutdown

To safely stop or reboot your cluster:

1. Stop the database.
2. Stop or reboot one or more instances.

Caution: If you stop or reboot an instance (or the cluster) without shutting the database down first, disk or database corruption could result. Shutting the database down first ensures that Vertica is not in the process of writing to disk when you shutdown. Refer to the Vertica Administrator's Guide for information on stopping a database.

Using Management Console (MC) on AWS

Management Console is a database management tool that allows you to view and manage aspects of your cluster. As of Vertica version 8.0.1, Micro Focus provides an MC AMI, which you can use with AWS.

The MC AMI allows you to create an instance, dedicated to running MC, that you can attach to a new or existing Verticacluster on AWS.

You can create and attach an MC instance to your Verticaon AWS cluster at any time, if your AWS cluster is running VerticaVertica version 8.0.1 or later.

Related Topics

- [Configuring and Launching an MC instance](#)
- [Network ACL settings](#)
- [Logging in to MC and Managing Your Cluster](#)

Configuring and Launching an MC instance

When you create an EC2 instance on AWS using a Vertica MC AMI, the instance includes a complete installation of Vertica MC. Hewlett Packard Enterprise recommends that you use the Vertica MC AMI as is – without modification.

configure and launch an MC instance in AWS:

1. Select the Vertica MC AMI from the AWS marketplace.
2. Create a single instance, Micro Focus International plc recommends c3.large for most implementations.
3. Place your MC instance within the VPC, subnet, and security group of the cluster which it will monitor.
4. Choose the key value pairs you use with Vertica.
5. Assign an elastic IP to the instance to make the MC externally accessible.

Configure Your Security Group:

The MC AMI requires specific security group configurations to communicate with the Vertica cluster. Enable the security rules for the MC AMI listed under the MC section of the Network ACL Settings section.

Launch MC Instance:

Verify that your MC instance is running.

Logging in to MC and Managing Your Cluster

After you launch your MC instance and configured your security group settings, log in to your database. To do so, use the elastic IP you specified during instance creation.

From this elastic IP, you can manage your Verticadatabase on AWS using standard MC procedures.

Considerations When Using MC on AWS

- Because MC is already installed on the MC AMI, the MC installation process does not apply.
- To uninstall MC on AWS, follow the procedures provided in [Uninstalling Management Console](#) before terminating the MC Instance.

Related Topics

- Refer to [Using Management Console](#) for information.
- Refer to [Managing Database Clusters](#) to create or import your cluster.

Adding Nodes to a Running AWS Cluster

Use these procedures to add instances/nodes to an AWS cluster. The procedures assume you have an AWS cluster up and running and have most-likely accomplished each of the following:

- Created a database.
- Defined a database schema.
- Loaded data.
- Run the database designer.
- Connected to your database.

Related Topics

- [Launching New Instances to Add to an Existing Cluster](#)
- [Including New Instances as Cluster Nodes](#)
- [Adding Nodes and Rebalancing the Database](#)

Launching New Instances to Add to an Existing Cluster

Perform the procedure in [Configuring and Launching an Instance](#) to create new instances that you then will add to your existing cluster. Be sure to choose the same details you chose when you created the original instances (VPC, placement group, subnet, and security group).

Including New Instances as Cluster Nodes

The **Instances** page lists the instances and their associated IP addresses. You need the IP addresses when you run the `install_vertica` script.

If you are configuring EBS volumes, be sure to configure the volumes on the node before you add the node to your cluster.

To add the new instances as nodes to your existing cluster:

1. [Configure and launch your new instances.](#)
2. Connect to the instance that is assigned to the Elastic IP. See [Connecting to an Instance](#) if you need more information.
3. Run the vertica installation script to add the new instances as nodes to your cluster. Specify the internal IP addresses for your instances and your `*pem` file name.

```
sudo /opt/vertica/sbin/install_vertica --add-hosts <instance-ip>--  
dba-user-password-disabled --point-to-point --data-dir  
/vertica/data --ssh-identity ~/<name-of-pem>.pem
```

Adding Nodes and Rebalancing the Database

Once you have added the new instances to your existing cluster, you add them as nodes to your cluster, and then rebalance the database.

Follow the procedure given in the Administration Guide, [Adding Nodes to a Database](#).

Removing Nodes From a Running AWS Cluster

Use these procedures to remove instances/nodes from an AWS cluster.

Related Topics

- [Preparing to Remove a Node](#)
- [Removing Hosts from the Database](#)
- [Removing Nodes from the Cluster](#)
- [Stopping the AWS instance](#)

Preparing to Remove a Node

Back up the Database. See the [Backup and Restore](#) sections of this document for more information.

Micro Focus recommends that you back up the database before performing this significant operation because it entails creating new projections, deleting old projections, and reloading data.

Removing Hosts From the Database

Before performing the procedure in this section, you must have completed the tasks referenced in [Preparing to Remove a Node](#). The following procedure assumes that you have both backed up your database and lowered the k-safety.

Note: Do not stop the database.

Perform the following to remove a host from the database.

1. While logged on as dbadmin, launch Administration Tools.

```
$ /opt/vertica/bin/admintools
```

Note: Do not remove the host that is attached to your EIP.

2. From the **Main Menu**, select **Advanced Tools Menu**.
3. From **Advanced Menu**, select **Cluster Management**. Select **OK**.
4. From **Cluster Management**, select **Remove Host(s)**. Select **OK**.
5. From **Select Database**, choose the database from which you plan to remove hosts. Select **OK**.
6. Select the host(s) to remove. Select **OK**.
7. Click **Yes** to confirm removal of the hosts.

Note: Enter a password if necessary. Leave blank if there is no password.

8. Select **OK**. The system displays a message letting you know that the hosts have been removed. Automatic re-balancing also occurs.
9. Select **OK** to confirm. Administration Tools brings you back to the **Cluster Management** menu.

Removing Nodes From the Cluster

To remove nodes from a cluster:

run the `install_vertica` script and specify:

- The option `--remove-hosts` followed by the IP addresses of the nodes you are removing
- The option `--ssh-identity` followed by the location and name of your `*pem` file. (The following example removes only one node from the cluster.)
- The option `--dba-user-password-disabled`

The following example removes 1 node from the cluster:

```
sudo /opt/vertica/sbin/install_vertica --remove-hosts 10.0.11.165 --  
point-to-point --ssh-identity ~/<name-of-pem>.pem --dba-user-password-  
disabled
```

Stopping the AWS Instances (Optional)

Once you have removed one or more nodes from your cluster, to save costs associated with running instances, you can choose to stop or terminate the AWS instances that were previously part of your cluster. This step is optional because, once you have removed the node from your Vertica cluster, Vertica no longer sees the instance/node as part of the cluster even though it is still running within AWS.

To stop an instance in AWS:

1. On AWS, navigate to your **Instances** page.
2. Right-click on the instance, and choose **Stop**.

Upgrading and Migrating Vertica on AWS

The topics in this section explain how you upgrade your version of Vertica and migrate data between clusters on AWS.

Upgrading to the latest Version of Vertica on AWS

Use these procedures for upgrading to the latest version of Vertica on AWS. The procedures assume that you have a 7.1 or later cluster successfully configured and running on AWS. If you are setting up an Vertica cluster on AWS for the first time, follow the [detailed procedure for installing and running Vertica on AWS](#).

Note: Both `install_vertica` and `update_vertica` use the same parameters.

Related Topics

- [Preparing to Upgrade Your Cluster](#)
- [Upgrading Vertica on AWS](#)

Preparing to Upgrade Your Cluster

Prepare for the upgrade to the latest Vertica version by performing the following:

1. Back up your existing database. See the [Backup and Restore](#) section for more information.
2. Download the Vertica install package. See [Download and Install the Vertica Install Package](#) in the Installation Guide for more information.

Upgrading Vertica Running on AWS

Vertica supports upgrades of Vertica Server running on AWS instances created from the Vertica AMI. To upgrade Vertica, follow the instructions provided in the Vertica [upgrade documentation](#).

Add the following arguments to the upgrade script:

- `--dba-user-password-disabled`
- `--point-to-point`

Migrating to Vertica 7.0 or later on AWS

Note: If you had a Vertica installation running on AWS prior to Release 6.1.x, you can migrate to Vertica 8.1.x or later using a new preconfigured AMI.

For more information, see the Solutions tab of the [myVertica portal](#).

Migrating Data Between AWS Clusters

This section provides guidance for copying (importing) data from another AWS cluster, or exporting data between AWS clusters.

There are common issues that occur when exporting or copying on AWS clusters. The issues are listed below. Except for these specific issues as they relate to AWS, copying and exporting data works as documented in the Administrator's Guide section, [Copying and Exporting Data](#).

1. **Ensure that all nodes in source and destination clusters have their own elastic IPs (or public IPs) assigned.**

If your destination cluster is located within the same VPC as your source cluster, proceed to step 3. Each node in one cluster must be able to communicate with each node in the other cluster. Thus, each source and destination node needs an elastic IP (or public IP) assigned.

2. **Set the parameter `DontCheckNetworkAddress` to true.**

On AWS, when creating a network interface, you receive an error if you attempt to assign the elastic IP to an AWS node (example uses a sample elastic IP address):

```
dbadmin=> CREATE NETWORK INTERFACE eipinterface ON v_tpch_node0001  
with '107.23.151.10';
```

ERROR 4125: No valid address found for [107.23.151.10] on this node
This error occurs because the elastic IP is the public IP and not the private IP of the target node. To resolve this issue, first set the parameter `DontCheckNetworkAddress` to true:

```
select set_config_parameter('DontCheckNetworkAddress', '1');
```

You can find information on the [CREATE NETWORK INTERFACE](#) statement and [SET_CONFIG_PARAMETER](#) in the SQL Reference Manual.

3. **Ensure your security group allows the AWS clusters to communicate.**

Check your security groups for both your source and destination AWS clusters. Ensure that ports 5433 and 5434 are open. If one of your AWS clusters is on a separate VPC, ensure that your network access control list (ACL) allows communication on port 5434.

Note: This communication method exports and copies (imports) data through the internet. You can alternatively use non-public IPs and gateways, or VPN to connect the source and destination clusters.

4. **If there is one or more ELB between the clusters, ensure that Port 5433 is opened between the ELBs and clusters.**
5. **If you use the Vertica client to connect to one or more ELBs, the ELBs only distribute incoming connections. The data transmission path occurs between clusters.**

Backup and Restore Vertica on Amazon Web Services

These sections outline three best-practice approaches for backing up and restoring Vertica clusters on Amazon Web Services (AWS) and the advance preparation required for each.

The examples in these sections recover a sample database, VMart, from multiple failures. To complete the steps in this guide, you should have a Vertica cluster on AWS with data on it.

More Information

For more information about backups, visit the [Backing Up and Restoring the Database](#) section of the Vertica documentation.

K-Safe Cluster Configuration

You can protect against isolated node failures with a K-safe cluster configuration. A K-safe cluster stores buddy data on other nodes in the cluster to prevent data loss if a node fails. For a cluster to be K-safe, it must consist of at least 3 nodes.

To learn more about K-safe cluster configurations, visit the [Designing For K-Safety](#) section of the Vertica documentation.

Recover from a K-Safe Failure

To recover from an isolated node failure, your cluster must still be in the UP state. Recovering from a K-safe failure does not require a backup.

If a node goes down and you cannot re-connect to it, you must re-create the node. Follow these steps to re-create a source node and recover from a K-safe failure:

- [Create a Target Node](#)
- [Recover a Node](#)

Note: If your primary node fails, you must reassign your Elastic IP, and then copy your .pem key file to one of your other running nodes to recover.

Create a Target Node

Create a target AWS instance ensuring the following:

- **Subnet, network, and VPC**—Your target instance must be in the same subnet, network, and VPC. It must also have the same network configurations as the nodes in your source cluster.
- **Cluster placement and availability zone**—Your target instance must be in the same cluster placement group and availability zone as the nodes in your source cluster.
- **IP address**—The internal IP address of your target instance must be the same as the IP address of the source node. Use the **Network Interfaces** option during instance creation to assign internal IP addresses.
- **Version compatibility**—Your target instance must use the same Vertica AMI version and hotfix version as the nodes in your source cluster.
- **Instance type**—Your target instance must use the same instance type as the nodes in your source cluster.

Recover a Node

1. On all your nodes, delete the .pem key information for the failed source node in the following locations:
 - /root/.ssh/known_hosts
 - /home/dbadmin/.ssh/known_hosts
2. On your main node, run the install script `install_vertica`, specifying:
 - your own key file
 - -Y option as point-to-point
 - dba user password disabled
 - dba user dbadmin:

```
sudo /opt/vertica/sbin/install_vertica -i ~/userkey.pem -Y --  
point-to-point --dba-user-password-disabled --dba-user dbadmin
```

3. Connect to your target node with SSH, and configure its storage to match your source cluster's storage.
4. Configure your target node:
 - a. Create an empty catalog and data directory matching the source node:

```
mkdir -p /vertica/data/VMart/v_vmart_node0003_catalog
```
 - b. Change the owner of the catalog and data directory to verticadba:

```
sudo chown dbadmin:verticadba /vertica/data/VMart
```
5. Restart the target node using admintools, specifying the IP address of your target node and your database name:

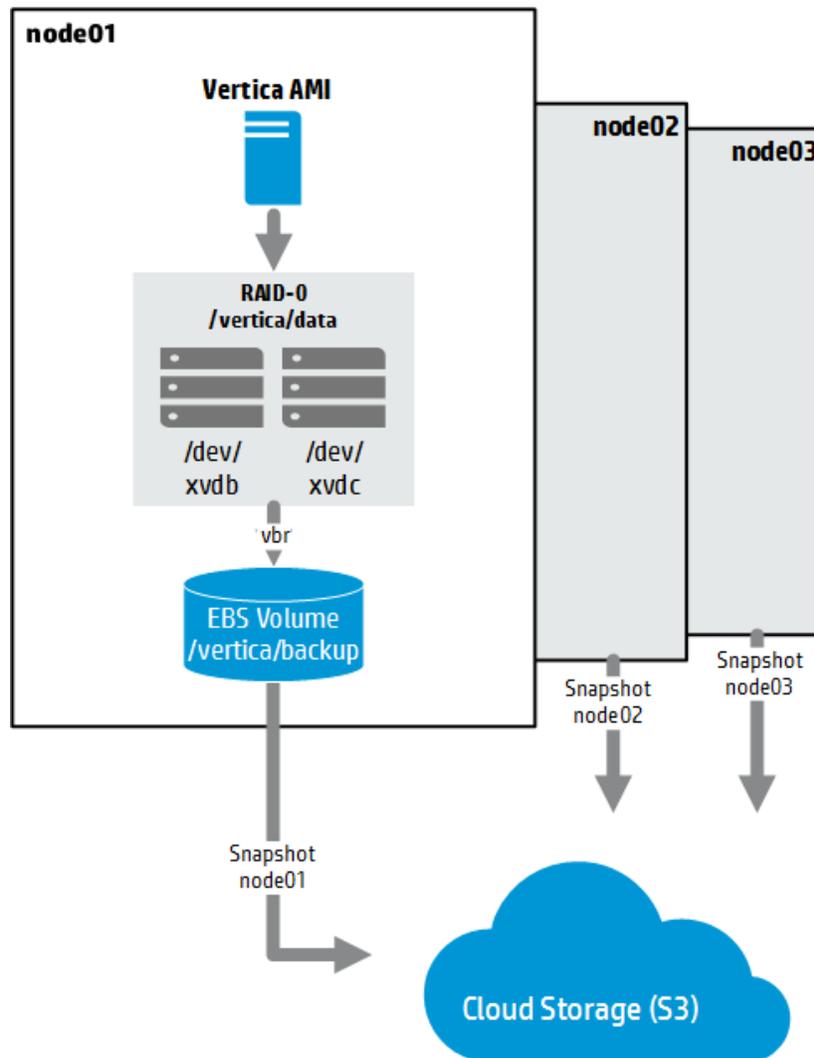
```
admintools -t restart_node -s 10.0.10.15 -d VMart
```

Your target node will now recover using data from its buddy, making your cluster K-safe once again. Depending on the size of your database, recovery may take some time.

Full Backup

A full backup captures a complete image of your database at a specific point in time. This option is the safest and most stable backup approach. A full backup lets you recover from a non K-safe loss, such as a multi-node failure or a total cluster failure.

Full Backup



Backup Artifacts

1. EBS snapshots (one for each node)
2. /opt/vertica/conf/admintools.conf
3. vmart_backup.ini

Storage Considerations

When you choose storage for your full backup, be aware of these considerations:

- If you are using ephemeral storage, you must perform a full backup.
- You must use EBS storage for the backup volume.

- To reduce costs, you may choose cheaper, slower EBS storage drive types for the backup volume.

To perform a full backup, you need the following:

- Properly formatted and mounted backup volumes
- An AWS snapshot of your backup volumes
- A backup configuration file

Find more information about full backups in the [Types of Backups](#) section of the Vertica documentation.

Related Topics

The process of creating a full backup on AWS requires the following tasks:

- [Prepare Backup Volumes](#)
- [Perform a Full Backup](#)

Prepare Backup Volumes

1. Find the size of the data catalog on each source node:

```
df -h /vertica/data/
```

Look for the number in the Used column.
2. Find the largest data catalog of all the nodes in your source cluster, and add a 20–50% safety margin.
3. Create and mount a new EBS volume to each source node. This volume should equal your largest data catalog with its safety margin.
4. Verify that your newly mounted volumes appear on each source node:

```
ls /dev
```
5. Format all volumes, using an Vertica supported file system:

```
sudo mkfs.ext4 /dev/xvdf
```
6. Create the backup folder. Specify the location where you mount the new backup volume:

```
sudo mkdir /vertica/backup
```

7. Mount the volume to `/vertica/backup`, and make it persistent:

```
sudo mount /vertica/backup  
sudo bash -c "echo '/dev/xvdf /vertica/backup ext4 defaults 0 0' >  
/etc/fstab"
```
8. Set `dbadmin` as the owner for all `/vertica/backup`:

```
sudo chown dbadmin:verticadba /vertica/backup
```
9. Verify the success of the mount operation by entering:

```
df -h
```
10. Repeat the formatting and mounting steps on all nodes.
11. Create a Backup configuration file by running `vbr` with the `--setupconfig` option:

```
/opt/vertica/bin/vbr --setupconfig
```

For more information about creating a backup configuration file, refer to [Creating vbr Configuration Files](#).
12. Once all of your backup locations exist, initialize those locations using the `vbr init` task:

```
vbr -t init -c config_file_name
```

Perform a Full Backup

1. Run `vbr`, specifying the `--config-file` and `--task backup` options:

```
/opt/vertica/bin/vbr --config-file vmart_backup.ini --task backup
```

Note: If your configuration file relies on a password file, you may need to copy your password file to each of your nodes to run `vbr`.
2. Verify that `vbr` completes without errors, and then make a snapshot of the backup volume on each of your nodes.
3. Save your `admintools` configuration file, which has information on your node IP addresses and mapping:

```
/opt/vertica/conf/admintools.conf
```

The snapshots, your backup configuration file, and your `admintools` configuration file make up your complete initial backup. Later in time, you may make an incremental backup by repeating steps 1 and 2.

Note:

- Incremental snapshots can take different amounts of time compared to the initial snapshot, because only the differences from the first snapshot are saved. If the differences are few, subsequent backups and snapshots can be quite fast.
- Snapshots are taken asynchronously. Therefore, you can continue to write the next backup as the previous snapshot completes.

For more information on EBS snapshots, visit the [AWS documentation](#).

See the [Creating Full and Incremental Backups](#) section of the Vertica documentation for additional information on incremental backups.

Restore from a Full Backup

If a cluster fails, use your backup configuration file and backup volume snapshots to restore your cluster from your last full backup.

Related Topics

The process of restoring from a full backup on AWS requires the following tasks:

- [Create a Target Cluster](#)
- [Restore a Database from a Full Backup](#)

Create a Target Cluster

Create a target cluster on AWS. Your new instances must match your source cluster in the following ways:

- **Network and VPC**—The target instances must be in the same network and VPC as each other. However, they do not have to be in the same network and VPC as your source cluster.
- **Number of instances and nodes**—You must use the same number of instances and nodes as your source cluster.

- **IP addresses**—The internal IP addresses of the target nodes must be the same as the internal IP addresses of the nodes in the source cluster.
- **Version compatibility**—Your target cluster must be running the same AMI and Vertica versions and using the same hotfix version as your source cluster.

Your new instances may differ from the source cluster in the following ways:

- Your target cluster may be in a different availability zone.
- You may use a different instance type for your target cluster.

Restore a Database from a Full Backup

After you create a target cluster, you must restore the database that was on the source cluster:

1. Using your source cluster backup snapshots, create and attach one volume for each node in your cluster and attach them to the nodes in your target cluster. Verify that the snapshots from the source cluster match with their respective nodes on the target cluster.

Important: You must use the correct device mapping. For example, a backup snapshot taken for node 1, must be recreated in the new cluster at node 1.

2. Mount the backup location on all nodes of the target cluster with the same file path as your source cluster:

```
sudo bash -c "echo '/dev/xvdf /vertica/backup ext4 defaults 0 0' > /etc/fstab"
sudo mkdir /vertica/backup
sudo mount /vertica/backup
```
3. Verify the success of your mounting operation by checking for data in your backup folder:

```
ls /vertica/backup/
```
4. Using your `admintools.conf` file backup as a reference, create an empty database on the target cluster with the same `dbadmin` username, password, data path, and database name as your source database.
5. Stop the database, if it is running.
6. Run a restore operation:

```
/opt/vertica/bin/vbr --config-file vmart_backup.ini --task restore
```
7. Start the database to conclude the restoration process.

Hard-Link Backup on AWS RAID-0

The way you perform a hard-link backup on AWS differs from doing a hard-link backup procedure on traditional bare metal Vertica installations. The use of a software RAID-0 device requires a different approach for AWS. Very small timing differences occur when taking the snapshot of the EBS volumes that make up RAID-0 devices. These timing differences can cause inconsistencies that make the backup invalid.

Important: Before performing a hard-link backup, you must freeze or unmount the RAID-0 file system.

You can choose between two options for performing a hard-link backup:

- Stop the cluster with `admintools`.
- Freeze the cluster with the `fsfreeze` command.

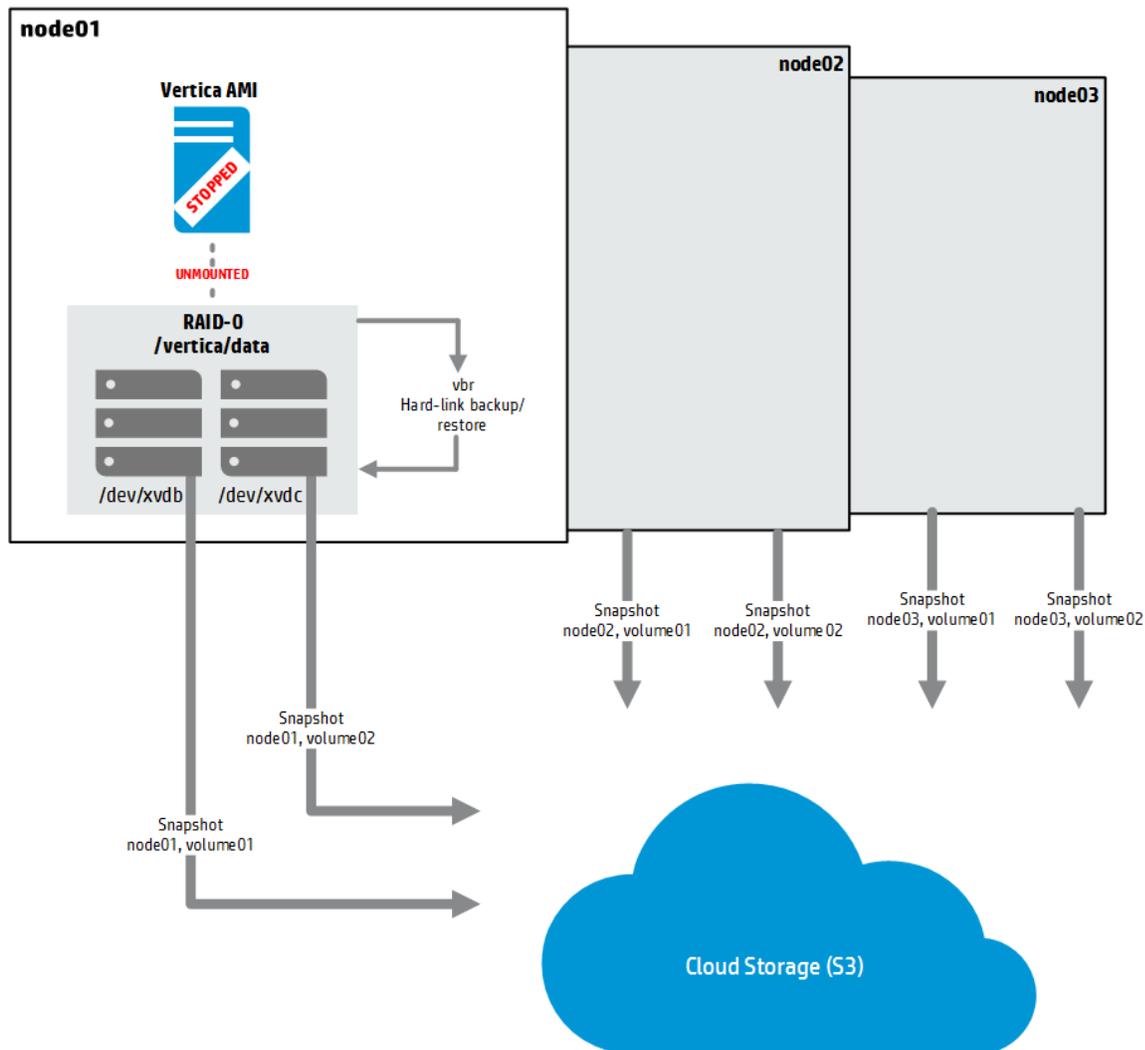
Note: You cannot perform a hard-link backup on AWS RAID-0 if your installation uses ephemeral volumes.

Hard-Link Backup with `admintools`

One way you can perform a hard-link backup is to stop the cluster with `admintools`. If you backup the volumes when the database is down, you can restore them without requiring you to run the `vbr` backup and restore script. However, if you want to maintain multiple point-in-time backups, you can still use `vbr`.

Use this approach if your service-level agreements allow you to stop the database for a period of time long enough to initiate snapshots.

Hard-Link Backup with admintools



Backup Artifacts

1. EBS snapshots (1 for each volume in RAID-0 on every cluster)
2. /opt/vertica/conf/admintools.conf
3. /etc/mdadm.conf (1 for each node)
4. vmart_backup.ini

Perform Hard-Link Backup Using admintools

Before performing this task, identify the instance IP addresses in your source cluster and make note of your RAID volumes for later use. Then, create a backup configuration file with hard-link backup enabled.

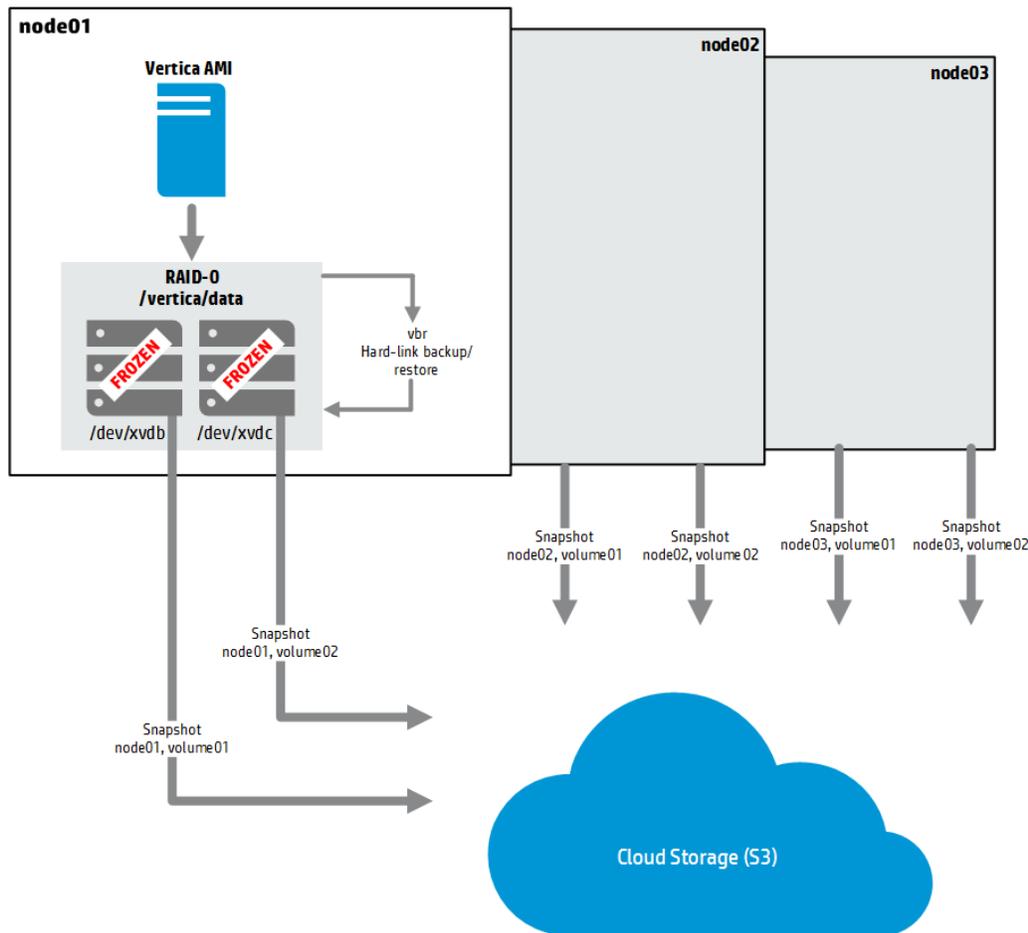
For information about enabling hard-link backup within a backup configuration file, see [Configuring the Hard Link Local Parameter](#) in the full Vertica documentation.

1. Stop the database:
`admintools -t stop_db -d VMart`
2. On each node, unmount your data volume:
`umount /vertica/data`
3. Create a snapshot of the RAID-0 volumes across your cluster, and make note of the each volume's corresponding snapshot ID. You need this information to assign snapshots to their correct node and volume designation during the restore process.
4. On each node, mount your data volume:
`mount /vertica/data`
5. Start your database:
`admintools -t start_db -d VMart`
6. Save the following:
 - Your RAID-0 configuration file for each node:
`/etc/mdadm.conf`
 - Your backup configuration file:
`vmart_backup.ini`
 - [Optional] Your admintools configuration file, which has information on your node IP addresses and mapping:
`/opt/vertica/conf/admintools.conf`

Hard-Link Backup with the fsfreeze Command

Another way you can perform a hard-link backup is freezing the cluster with the `fsfreeze` command. Because you can perform this kind of backup without stopping the cluster, users experience minimal performance effects. You must use `vbr` to restore from a backup performed using the `fsfreeze` command.

Hard-Link Backup with fsfreeze Command



- Backup Artifacts**
1. EBS snapshots (1 for each volume in RAID-0 on every cluster)
 2. /opt/vertica/conf/admintools.conf
 3. /etc/mdadm.conf (1 for each node)
 4. vmart_backup.ini

Perform a Hard-Link Backup with the fsfreeze Command

Before you can perform this procedure, you must identify the instance IP addresses in your source cluster, and make note of your RAID volumes. You also must create a backup configuration file with hard-link backup enabled. For information about enabling hard-link backup within a backup configuration file, see [Configuring the Hard Link Local Parameter](#) in the full Vertica a documentation.

1. Make a hard link backup on the RAID-0 device.
`/opt/vertica/bin/vbr --config-file vmart_backup.ini --task backup`

2. Freeze the RAID-0 volume across the cluster for a consistent snapshot of the EBS volumes that constitute that device. Freezing halts *all* database/SQL operation until you unfreeze the volume.

Important: Do not create a snapshot of a RAID-0 volume without freezing it first. Performing a snapshot without freezing your RAID volume invalidates your snapshot. Always check the return code of the `fsfreeze` command to ensure the device is frozen before you proceed.

```
for IP in 10.0.10.13 10.0.10.14 10.0.10.15; do ssh $IP sudo  
fsfreeze --freeze /vertica/data;
```

3. Create a snapshot of the RAID-0 volumes across your cluster. Make note of the each volume's corresponding snapshot ID. You will need this information to assign snapshots to their correct volume designation during the restore process.
4. After the snapshot has started for all EBS volumes on all nodes, unfreeze the file system:
for IP in 10.0.10.13 10.0.10.14 10.0.10.15; do ssh \$IP sudo
fsfreeze --unfreeze /vertica/data;

Note: You do not need to wait for the snapshot to complete before unfreezing.

5. Save the following:
 - Your RAID-0 configuration file for each node:
`/etc/mdadm.conf`
 - Your backup configuration file:
`vmart_backup.ini`
 - [Optional] Your admintools configuration file, which has information on your node IP addresses and mapping:
`/opt/vertica/conf/admintools.conf`

Restore from a Hard-Link Backup

If a cluster fails, use your backup configuration file and backup volume snapshots to restore your cluster from your last hard-link backup.

Related Topics

The process of restoring from a hard-link backup on AWS requires the following tasks:

- [Create a Target Cluster](#)
- [Restore a Database from a Hard-Link Backup](#)

Create a Target Cluster

Create a target cluster on AWS. Your new instances must match your source cluster in the following ways:

- **Network and VPC**—The target instances must be in the same network and VPC as each other. However, they do not have to be in the same network and VPC as your source cluster.
- **Number of instances and nodes**—You must use the same number of instances and nodes as your source cluster.
- **IP addresses**—The internal IP addresses of the target nodes must be the same as the internal IP addresses of the nodes in the source cluster.
- **Version compatibility**—Your target cluster must be running the same AMI and Vertica versions and using the same hotfix version as your source cluster.

Your new instances may differ from the source cluster in the following ways:

- Your target cluster may be in a different availability zone.
- You may use a different instance type for your target cluster.

Restore a Database from a Hard-Link Backup

1. Create EBS volumes from your backup snapshots.

Note: Before you attach your new volumes, stop and detach any existing RAID volumes mounted to the target cluster.

2. Attach the new EBS volumes to their respective nodes and volumes on the target cluster.

Important: You must use the correct device mapping. For example, a backup snapshot taken for node 1, volume `/dev/xvdf` must be re-created in the new cluster at node 1, volume `/dev/xvdf`.

3. Rebuild your RAID-0 device by restoring your RAID configuration file:
`/etc/mdadm.conf`
4. When the volumes have finished attaching, remount the RAID:
`for IP in 10.0.10.13 10.0.10.14 10.0.10.15; do ssh $IP sudo mount /vertica/data ;`
5. Create an empty database on the target cluster with the same dbadmin username, password, datapath, and database name as your source database.
6. Run a restore operation:
`/opt/vertica/bin/vbr --config-file vmart_backup.ini --task restore`

Troubleshooting Vertica On AWS

The following sections contain information for troubleshooting Vertica on AWS.

Related Topics

- [Checking Open Ports Manually Using the Netcat Utility](#)

Checking Open Ports Manually Using the Netcat Utility

Once your cluster is up and running, you can check ports manually through the command line using the netcat (nc) utility. What follows is an example using the utility to check ports.

Before performing the procedure, choose the private IP addresses of two nodes in your cluster.

The examples given below use nodes with the private IPs:

```
10.0.11.60 10.0.11.61
```

Install the nc utility on your nodes. Once installed, you can issue commands to check the ports on one node from another node.

1. To check a TCP port:
 - a. Put one node in listen mode and specify the port. In the following sample, we're putting IP `10.0.11.60` into listen mode for port `4804`.

```
[root@ip-10-0-11-60 ~]# nc -l 4804
```

- b. From the other node, run nc specifying the IP address of the node you just put in listen mode, and the same port number.

```
[root@ip-10-0-11-61 ~]# nc 10.0.11.60 4804
```

- c. Enter sample text from either node and it should show up on the other. To cancel once you have checked a port, enter **Ctrl+C**.

Note: To check a UDP port, use the same nc commands with the `-u` option.

```
[root@ip-10-0-11-60 ~]# nc -u -l 4804  
[root@ip-10-0-11-61 ~]# nc -u 10.0.11.60 4804
```

Vertica on Microsoft Azure

Welcome to the Vertica on Microsoft Azure guide. This section explains how you can create Vertica clusters on the Microsoft Azure Cloud Platform.

Overview of Vertica on Microsoft Azure

Vertica clusters on Microsoft Azure operate on virtual machines (VMs) within a virtual network. The instructions in this document apply to VMs in Azure that are built with Vertica.

For more information about Azure VMs, see the [Azure documentation](#).

Recommended VM Types

Vertica supports a range of Microsoft Azure virtual machine (VM) types, each optimized for different purposes. Choose the VM type that best matches your performance and price needs as a user.

For the best performance in most common scenarios, use one of the following VMs:

Optimization	Type
Memory optimized	DS13_V2 DS14_V2 DS15_V2
High memory and I/O throughput	GS3 GS4 GS5

Note: The GS VMs are not available in all regions. Nor are they available on the Azure Marketplace, but you can use them by following

the manual
deployment steps
described in [Deploy
Vertica on
Azure: Manual Steps](#).

Note: Data stored on the drives of a VM exists only while that VM is powered on. After powering off a VM, data on temporary drives is lost.

Supported Azure Operating Systems

For best performance, use one of the following operating systems when deploying Vertica on Azure:

- Red Hat 7.3
- CentOS 7.3. The Azure Marketplace solution as of this writing (June 2017) is based on CentOS 7.3.1611.

For more information, see [Vertica Supported Platforms](#).

Deploy Vertica from the Microsoft Azure Marketplace

For customers looking to quickly deploy a Vertica cluster in the Microsoft Azure Cloud, this section describes an automated deployment solution that can be accessed from the Azure Marketplace at [Vertica Analytics Platform](#).

To deploy Vertica on Microsoft Azure, you need to upgrade from the Free Trial, which limits you to a 4-core quota, to Microsoft Azure's Pay-As-You-Go subscription. The Pay-As-You-Go subscription requires a minimum quota of 8 cores which is what is needed for the Vertica Marketplace solution for Azure.

Check with Microsoft about the benefits and credits that you can carry over from the Free Trial to the Pay-As-You-Go subscription. For more information about upgrading a free Microsoft Azure trial account to Pay-As-You-Go, see [Azure pricing](#).

When you're ready, click **Get It Now** on [Vertica Analytics Platform](#). The following sections describe the steps you need to take after you click that button.

Basic Information that Azure Needs

On the first plane (window) of the installer for your Vertica-Azure solution, enter the following information:

- **DBADMIN username**—The primary username for the virtual machines being created. This name is also used as the Vertica DBADMIN account.
- **Authentication Type**—VMs in Azure support either password or SSH public key authentication.
- **Password or SSH public key**—Enter either a password or a public key to be associated with the DBADMIN user.
- **Root password**—Enter a password to be assigned to the root user. This is used during the Vertica installation.
- **Subscription**—Select an Azure subscription for the resources to be billed against.
- **Resource group**—Create a new resource group or select an existing resource group as the location to save all the Azure resources for your Vertica cluster.
- **Location**—Select an Azure data center to deploy the Vertica cluster. Make sure that you have enough quota assigned to that location.

Infrastructure Settings

In the second plane of the installer, make choices relating to the VMs and the virtual infrastructure.

- **Virtual machine size**—Vertica engineers have designated the three different VM types for deployment that are available on the Azure Marketplace. When you highlight this field, Azure displays those types and their specific characteristics.

- **Number of cluster nodes**—This drop-down field allows you to choose the number of nodes for Azure to deploy for the cluster. The Marketplace solution allows you to create between 1 and 15 nodes.

Important: To maintain K-safety for your database, select at least three nodes.

- **Total RAW storage per node**—Each VM comes configured with a premium data disk for Vertica to use. The choices in this drop-down are 768GB, 3TB, or 6TB.
- **Storage account name prefix**—All disks associated with the cluster are contained in a group of four premium storage accounts. Storage accounts must each have a unique name. When Azure creates the storage accounts, the Marketplace appends a number to the end of each name.
- **Storage account type**—Because Vertica is a high-end enterprise solution that requires fast, low-latency storage, Premium-LRS storage is the only available choice for this field.
- **Virtual network**—The Azure Marketplace solution for Vertica allows you to create a new virtual network, or select an existing virtual network, in which to place the Vertica cluster. This control on the deployment provides a point-and-click navigation to do just that.
- **Subnets**—Based on the selection of the virtual network, the Subnets entry allows you to select or create the subnet you want use in the virtual network.
- **Public IP address**—Each VM is configured with a publicly accessible IP address. This field allows you to specify the resource name for those IP addresses, and whether they are static or dynamic. When created, Azure appends a number from 1 to 15 to each resource name. This number designates which VM that the resource is associated with.
- **Domain name label**—Because each VM has a public IP address, each node also needs a DNS name. Enter a prefix for the name, such as `vertnode`. Azure appends a number from 1 to 15 to the DNS name. That number designates which VM it is associated with. Azure adds the remaining part of the fully qualified domain name based on the location where you created the cluster.

Vertica Settings for Azure

Vertica engineers have designated the third installer plane for additional options to be included in the Vertica cluster installation. These options are:

- **Include Vertica Management Console**—If you select YES, the marketplace installer includes the deployment of a smaller VM to be configured as a Vertica Management Console and placed in the same network as the Vertica cluster.
- **Virtual machine size**—This is the virtual machine size for the node that hosts Management Console. Vertica engineers have designated three different VM types for deployment that are available from the Azure Marketplace to support the Vertica Management Console. If you highlight this field, you can access detailed information about each VM type.

Validate Your Configuration

Review the settings on the installer Summary plane. Azure uses this plane to validate the values you entered.

In addition, Azure checks your settings against the existing quota to make sure that your solution deploys correctly. After you are satisfied with your choices and Azure has notified you that the validation passed, click **OK**.

Finalize the Purchase

On the final plane of the installer, agree to the Microsoft Terms of Use, and acknowledge that you agree to pay for the resources being created.

In addition, on this plane, review the Vertica Terms of Use and the privacy policy. After you click **Purchase**, the deployment begins.

Monitor the Deployment

After the deployment begins, you receive a notification that the deployment is in progress. When you click the notification, the Azure portal opens the deployment status page, which displays the resources for the cluster being created.

Access the Cluster After Deployment

After the deployment successfully completes, you receive a notification in the Azure portal. When you click the notification, the Azure portal opens the resource group and displays information about all the resources created.

Take the following steps to finalize the cluster installation:

1. Select the public IP address name of any Vertica cluster node to view the values for that node. Copy either the IP address or the DNS name for that node, and switch to an SSH client.
2. Connect to the node using the DBADMIN user you created along with the authentication method you chose (public key or password).
3. Once logged on to the VM as the DBADMIN user, to start the Vertica admintools application, enter the following command:

```
[dbadmin@vertnode1 ~]$ admintools
```

4. When asked, do one of the following:
 - Provide a path to a Vertica License File, or
 - Press **Enter** to accept the Vertica Community Edition.
5. And finally, accept the End User License Agreement (EULA).

At this point, your cluster is fully installed and you are ready to create a new database.

Access Management Console After Deployment

After the deployment successfully completes, you receive a notification in the Azure portal. When you click the notification, the Azure portal opens the resource group and displays information about all the resources created.

Take these steps to launch Management Console:

1. Select the public IP address name for the Management Console cluster. The name starts with VMC and is concatenated with the public IP address name you entered during installation.
2. Copy either the IP address or the DNS name for the node, and switch to a web browser.
3. Enter the following into your browser navigation field. This command launches the Management Console:

```
https://[IP address or DNS name]:5450
```

Now you can complete the installation process.

For complete information about how to use the Vertica Management Console, see [Using Management Console](#) in the Vertica documentation.

Deploy Vertica on Azure: Manual Steps

To start creating your Vertica cluster in Azure using manual steps, you first need to create a VM. During the VM creation process, you create and configure the other resources required for your cluster, which are then available for any additional VMs that you create.

Follow these procedures to deploy Vertica on Azure.

Configuring and Launching a New Instance

An Azure VM is similar to a traditional host. Just as with an on-premises cluster, you must prepare and configure the hardware settings for your cluster and network before you install Vertica.

The first steps are:

1. From the Azure marketplace, select an operating system that Vertica supports.
2. Select a VM type. See [Recommended VM Types](#)
3. Choose a deployment model. For best results, choose the resource manager deployment model.

Configure Network Security Group

Vertica has specific network security group requirements, as described in [Network Security Group Configurations](#).

Create and name your own network security group, following these guidelines.

You must configure SSH as:

- Protocol: TCP
- Source port range: Any
- Destination port range: 22

- Source: Any
- Destination: Any

You can make additional modifications, based on your specific requirements.

Add Disk Containers

Create an Azure storage account, which later contains your cluster storage disk containers.

For optimal throughput, select Premium storage and align the storage to a GS or DSv2 VM.

For more information about what a storage account is, and how to create one, refer to [About Azure storage accounts](#).

Configure Credentials

Create a password or assign an SSH key pair to use with Vertica.

For information about how to use key pairs in Azure, see [How to create and use an SSH public and private key pair for Linux VMs in Azure](#).

Assign a Public IP Address

A public IP is an IP address that you can use to connect to your cluster externally. For best results, assign a single static public IP to a node in your cluster. You can then connect to other nodes in your cluster from your primary node using the internal IP addresses that Azure generated when you specified your virtual network settings.

By default, a public IP address is dynamic; it changes every time you shut down the server. You can choose a static IP address, but doing so can add cost to your deployment.

During a VM installation, you cannot set a DNS name. If you use dynamic public IPs, set the DNS name in the public IP resource for each VM after deployment.

For information about public IP addresses, refer to [IP address types and allocation methods in Azure](#).

Create Additional VMs

If needed, to create additional VMs, repeat the previous instructions in this document.

Connect to a Virtual Machine

Before you can connect to any of the VMs you created, you must first make your virtual network externally accessible. To do so, you must attach the public IP address you created during network configuration to one of your VMs.

Connect to Your VM

To connect to your VM, complete the following tasks:

1. Connect to your VM using SSH with the public IP address you created in the configuration steps.
2. Authenticate using the credentials and authentication method you specified during the VM creation process.

Connect to Other VMs

Connect to other virtual machines in your virtual network by first using SSH to connect to your publicly connected VM. Then, use SSH again from that VM to connect through the private IP addresses of your other VMs.

If you are using private key authentication, you may need to move your key file to the root directory of your publicly connected VM. Then, use PuTTY or WinSCP to connect to other VMs in your virtual network.

Prepare the Virtual Machines

After you create your VMs, you need to prepare them for cluster formation.

Add the Vertica License and Private Key

Prepare your nodes by adding your private key (if you are using one) to each node and to your Vertica license. These steps assume that the initial user you configured is the DBADMIN user.

1. As the DBADMIN user, copy your private key file from where you saved it locally onto your primary node.

Depending upon the procedure you use to copy the file, the permissions on the file may change. If permissions change, the `install_vertica` script fails with a message similar to the following:

```
Failed Login Validation 10.0.2.158, cannot resolve or connect to host as root.
```

If you receive a failure message, enter the following command to correct permissions on your private key file:

```
$ chmod 600 /<name-of-key>.pem
```

2. Copy your Vertica license to your primary VM. Save it in your home directory or other known location.

Install Software Dependencies for Vertica on Azure

In addition to the Vertica standard [Package Dependencies](#), as the root user, you must install the following packages before you install Vertica on Azure:

- pstack
- mcelog
- sysstat
- dialog (Required for Vertica 8.0.x)

Configure Storage

Use a dedicated Azure storage account for node storage.

Caution: Do *not* store your information on the root volume, especially your data and catalog directories. Storing information on the root volume may result in data loss.

When configuring your storage, make sure to use a [supported file system](#).

Attach Disk Containers to Virtual Machines (VMs)

Using your previously created storage account, attach disk containers to your VMs that are appropriate to your needs.

For best performance, combine multiple storage volumes into RAID-0. For most RAID-0 implementations, attach 6 storage disk containers per VM.

Combine Disk Containers for Storage

If you are using RAID, follow these steps to create a RAID-0 drive on your VMs. The following example shows how you can create a RAID-0 volume named `md10` from 6 individual volumes named:

- `sdc`
- `sdd`
- `sde`
- `sdf`
- `sdg`
- `sdh`

1. Form a RAID-0 volume using the `mdadm` utility:

```
$ mdadm --create /dev/md10 --level 0 --raid-devices=6  
/dev/sdc /dev/sdd /dev/sde /dev/sdf /dev/sdg /dev/sdh
```

2. Format the file system to be one that Vertica supports:

```
$ mkfs.ext4 /dev/md10
```

3. Find the UUID on the newly formed RAID volume:

```
ls -l /dev/disk/by-uuid
```

4. Copy the UUID and place it in the FSTAB file.

```
$ sudo vi /etc/fstab  
UUID=<uuid here> /dev/md10 ext4 defaults , errors=remount-ro 1
```

5. Create folders for your Vertica data and catalog.

```
$ mkdir /vertica  
$ mkdir /vertica/data
```

6. Mount the RAID volume to your data and catalog directories.

```
$ mount /dev/md10 /home/dbadmin/vertica/data
```

Create a Swap File

In addition to storage volumes to store your data, Vertica requires a swap volume or swap file to operate.

Create a swap file or swap volume of at least 2 GB. The following steps show how to create a swap file within Vertica on Azure:

1. Install devnull and swapfile:

```
$ install -o root -g root -m 0600 /dev/null /swapfile
```

2. Create the swap file:

```
$ dd if=/dev/zero of=/swapfile bs=1024 count=2048k
```

3. Prepare the swap file using mkswap:

```
$ mkswap /swapfile
```

4. Use swapon to instruct Linux to swap on the swap file:

```
$ swapon /swapfile
```

5. Persist the swapfile in FSTAB:

```
$ echo "/swapfile          swap          swap          auto          0          0" >> /etc/fstab
```

Repeat the volume attachment, combination, and swap file creation procedures for each VM in your cluster.

For More Information

- [About Azure storage accounts](#)
- [Prepare Disk Storage Locations](#)

Download Vertica

To download the Vertica server appropriate for your operating system and license type, go to my.vertica.com/download/vertica.

Run the rpm to extract the files.

After you complete the download and extraction, the next section describes how to use the `install_vertica` script to form a cluster and install the Vertica database software.

Form a Cluster and Install Vertica

Use the `install_vertica` script to combine two or more individual VMs to form a cluster and install the Vertica database.

Before You Start

Before you run the `install_vertica` script:

- Check the **Virtual Network** page for a list of current VMs and their associated private IP addresses.
- Identify your storage location. The installer assumes that you have mounted your storage to `/vertica/data`. To specify another location, use the `--data-dir` argument.
- Identify your storage location. To create your database's data directory on mounted RAID drive, when you run the `install_vertica` script, provide `/vertica/data` as the value of the `--data-dir` option .

Caution: Do *not* store your data on the root drive.

Combine Virtual Machines (VMs)

The following example shows how to combine VMs using the `install_vertica` script.

1. While connected to your primary node, construct the following command to combine your nodes into a cluster.

```
$ sudo /opt/vertica/sbin/install_vertica --hosts 10.2.0.164,10.2.0.165,10.2.0.166 --dba-user-  
password-disabled --point-to-point --data-dir /vertica/data --ssh-identity ~/<name-of-private-  
key>.pem --license <license.file>
```

2. Substitute the IP addresses for your VMs and include your root key file name, if applicable.
3. Include the `--point-to-point` parameter to configure spread to use direct point-to-point communication between all Vertica nodes, as required for clusters on Azure when installing or updating Vertica.
4. If you are using Vertica Community Edition, which limits you to three nodes, specify `-L CE` with no license file.
5. After you combine your nodes, to reduce security risks, keep your key file in a secure place—separate from your cluster—and delete your on-cluster key with the `shred` command:

```
$ shred examplekey.pem
```

Important: You need your key file to perform future Vertica updates.

6. Reboot your cluster to complete the cluster formation and Vertica installation.

For complete information on the `install_vertica` script and its parameters, see [Installing Vertica with the Installation Script](#).

After Your Cluster is Up and Running

Now that your cluster is configured and running, and Vertica is running, take these steps:

1. [Creating a Database](#). When you installed Vertica, a database administrator user was created: DBADMIN.
2. Use this account to create and start a database. For detailed information about the DBADMIN role, see [DBADMIN Role](#).
3. [Configuring the Database](#). After the database is installed, it's important to configure its settings, schemas, and security.

Using Management Console (MC) on Azure

The Vertica solution in the Azure marketplace provides a Management Console (MC) virtual machine. The MC VM allows you to deploy a VM dedicated to running MC onto a Vertica cluster on Azure.

You can deploy an MC VM to your Vertica on Azure cluster at any time, as long as your cluster meets the following requirements:

- Your Azure cluster must be running Vertica version 8.0.x or later.
- Your MC version must be compatible with the Vertica version of your cluster.

Deploy an MC VM

To deploy MC onto your target Vertica cluster on Azure, perform these tasks:

1. Select the Vertica Management Console deployment from the Azure Marketplace.
2. Specify an admin username for your new MC VM.
3. Place your MC VM in the same location as your target cluster.
4. Place your MC VM in a resource group, separate from your target cluster.
5. Specify the virtual network and subnet of your target cluster.
6. Specify a DNS name and public IP address.

Configure the Network Security Group for the MC VM

The MC VM requires specific security group configurations in order to communicate with the Vertica cluster. Configure the security rules for the MC VM as described in the section "Configure Network Security Group" in [Configure Your Virtual Machine](#).

Log in to MC and Manage Your Cluster

After you have deployed your MC VM and configured your security group settings, enter the following into your browser navigation field. This command launches the Management Console:

```
https://[IP address or DNS name]:5450
```

Log in to Management Console using the public IP address that you specified during instance creation.

From there, you can manage your Vertica database on Azure using standard MC procedures.

Uninstalling MC

To uninstall MC on Azure, follow the procedures provided in [Uninstalling Management Console](#) before terminating the MC node.

Related Topics

- [Using Management Console](#)
- [Managing Database Clusters](#)

Deploy the Vertica Test Drive for Azure

For users looking to experience Vertica in a “test drive” environment on Azure, we offer that capability at the Azure Marketplace at [Vertica Analytics Platform](#).

When you click the **Test Drive** button, Azure creates a single Vertica node that is preloaded with clickstream and sales data. This Test Drive allows you to explore and analyze the data before and after conducting A/B testing.

After the deployment completes, you receive an email with instructions on how to access the Test Drive. You can also consult the Test Drive user manual: [Vertica Test Drive for Clickstream Analytics](#).

Integrating with Apache Hadoop

Apache™ Hadoop™, like Vertica, uses a cluster of nodes for distributed processing. The primary component of interest is HDFS, the Hadoop Distributed File System.

You can use Vertica with HDFS in several ways:

- You can import HDFS data into locally-stored ROS files .
- You can access HDFS data in place using external tables. You can define the tables yourself or get schema information from HCatalog, a Hadoop component.
- You can use HDFS as a storage location for ROS files.
- You can export data from Vertica to share with other Hadoop components using a Hadoop columnar format.

See [Hadoop Interfaces](#) for more information about these options.

A Hadoop cluster can use Kerberos authentication to protect data stored in HDFS. Vertica integrates with Kerberos to access HDFS data if needed. See [Using Kerberos with Hadoop](#).

Hadoop Distributions

Vertica can be used with Hadoop distributions from Hortonworks, Cloudera, and MapR. See [Vertica Integrations for Hadoop](#) for the specific versions that are supported.

If you are using Cloudera, you can manage your Vertica cluster using Cloudera Manager. See [Integrating With Cloudera Manager](#).

If you are using MapR, see [Integrating Vertica with the MapR Distribution of Hadoop](#).

Cluster Architecture

Vertica supports two cluster architectures. Which architecture you use affects the decisions you make about integration. These options might also be limited by license terms.

- You can co-locate Vertica on some or all of your Hadoop nodes. Vertica can then take advantage of data locality.
- You can build a Vertica cluster that is separate from your Hadoop cluster. In this configuration, Vertica can fully use each of its nodes; it does not share resources with Hadoop.

These layout options are described in [Cluster Layout](#).

File Paths

Hadoop file paths are expressed as URLs in the `hdfs` or `webhdfs` URL scheme. If you need to escape a special character in a path, use URL escaping. All input characters that are not a-z, A-Z, 0-9, '-', '.', '_' or '~' must be converted to URL encoding (%NN where NN is a two-digit hexadecimal number). The following example URL-encodes a file name with a space in it.

```
hdfs:///opt/data/my%20file.orc
```

You can use globs, including regular expressions, in file paths. When Hive writes data, it sometimes creates temporary files with a "`_COPYING`" suffix. If you try to read these files into Vertica you will get an error message, because they are not a valid format. The following example copies only files ending in digits, the usual format for exports from Hive:

```
=> CREATE EXTERNAL TABLE (...) AS COPY FROM hdfs:///data/parquet/*_[0-9] PARQUET;
```

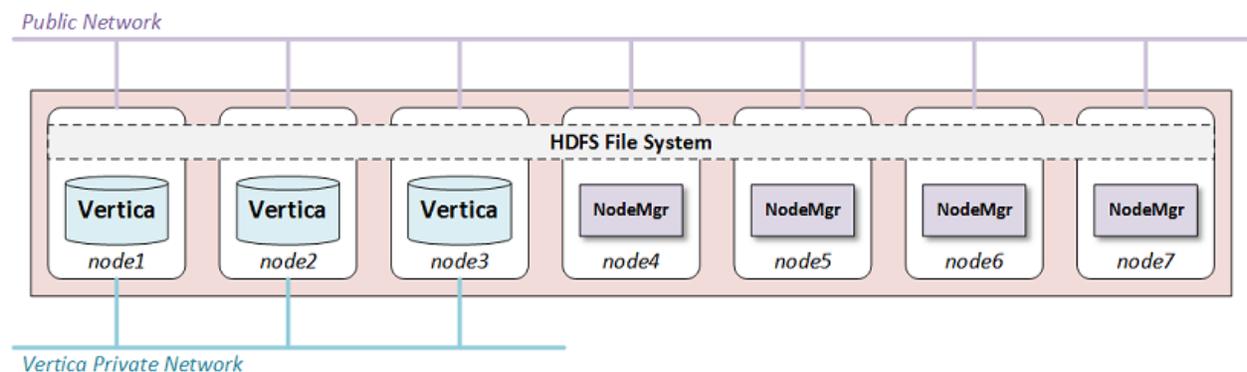
Cluster Layout

Vertica and Hadoop each use a cluster of nodes for distributed processing. These clusters can be co-located, meaning you run both products on the same machines, or separate. See [Co-located Clusters](#) and [Separate Clusters](#).

With either architecture, if you are using the `hdfs` scheme to read ORC or Parquet files, you must do some additional configuration. See [Configuring the hdfs Scheme](#).

Co-located Clusters

With co-located clusters, Vertica is installed on some or all of your Hadoop nodes. The Vertica nodes use a private network in addition to the public network used by all Hadoop nodes, as the following figure shows:



You might choose to place Vertica on all of your Hadoop nodes or only on some of them. If you are using HDFS Storage Locations you should use at least three Vertica nodes, the minimum number for [K-Safety](#).

Using more Vertica nodes can improve performance because the HDFS data needed by a query is more likely to be local to the node.

You can place Hadoop and Vertica clusters within a single rack, or you can span across many racks and nodes. If you do not co-locate Vertica on every node, you can improve performance by co-locating it on at least one node in each rack. See [Configuring Rack Locality](#).

Normally, both Hadoop and Vertica use the entire node. Because this configuration uses shared nodes, you must address potential resource contention in your configuration on those nodes. See [Configuring Hadoop for Co-located Clusters](#) for more information. No changes are needed on Hadoop-only nodes.

Hardware Recommendations

Hadoop clusters frequently do not have identical provisioning requirements or hardware configurations. However, Vertica nodes should be equivalent in size and capability, per the best-practice standards recommended in [General Hardware and OS Requirements and Recommendations](#) in Installing Vertica.

Because Hadoop cluster specifications do not always meet these standards, Micro Focus recommends the following specifications for Vertica nodes in your Hadoop cluster.

Specifications For...	Recommendation
Processor	<p>For best performance, run:</p> <ul style="list-style-type: none">• Two-socket servers with 8–14 core CPUs, clocked at or above 2.6 GHz for clusters over 10 TB• Single-socket servers with 8–12 cores clocked at or above 2.6 GHz for clusters under 10 TB
Memory	<p>Distribute the memory appropriately across all memory channels in the server:</p> <ul style="list-style-type: none">• Minimum—8 GB of memory per physical CPU core in the server• High-performance applications—12–16 GB of memory per physical core• Type—at least DDR3-1600, preferably DDR3-1866
Storage	<p>Read/write:</p> <ul style="list-style-type: none">• Minimum—40 MB/s per physical core of the CPU• For best performance—60–80 MB/s per physical core <p>Storage post RAID: Each node should have 1–9 TB. For a production setting, Micro Focus recommends RAID 10. In some cases, RAID 50 is acceptable.</p> <p>Because Vertica performs heavy compression and encoding, SSDs are not required. In most cases, a RAID of more, less-expensive HDDs performs just as well as a RAID of fewer SSDs.</p>

	If you intend to use RAID 50 for your data partition, you should keep a spare node in every rack, allowing for manual failover of a Vertica node in the case of a drive failure. A Vertica node recovery is faster than a RAID 50 rebuild. Also, be sure to never put more than 10 TB compressed on any node, to keep node recovery times at an acceptable rate.
Network	10 GB networking in almost every case. With the introduction of 10 GB over cat6a (Ethernet), the cost difference is minimal.

Configuring Hadoop for Co-Located Clusters

If you are co-locating Vertica on any HDFS nodes, there are some additional configuration requirements.

Hadoop Configuration Parameters

For best performance, set the following parameters with the specified minimum values:

Parameter	Minimum Value
HDFS block size	512MB
Namenode Java Heap	1GB
Datanode Java Heap	2GB

WebHDFS

Hadoop has two services that can provide web access to HDFS:

- WebHDFS
- httpFS

For Vertica, you must use the WebHDFS service.

YARN

The YARN service is available in newer releases of Hadoop. It performs resource management for Hadoop clusters. When co-locating Vertica on YARN-managed Hadoop nodes you must make some changes in YARN.

Micro Focus recommends reserving at least 16GB of memory for Vertica on shared nodes. Reserving more will improve performance. How you do this depends on your Hadoop distribution:

- If you are using Hortonworks, create a "Vertica" node label and assign this to the nodes that are running Vertica.
- If you are using Cloudera, enable and configure static service pools.

Consult the documentation for your Hadoop distribution for details. Alternatively, you can disable YARN on the shared nodes.

Hadoop Balancer

The Hadoop Balancer can redistribute data blocks across HDFS. For many Hadoop services, this feature is useful. However, for Vertica this can reduce performance under some conditions.

If you are using HDFS storage locations, the Hadoop load balancer can move data away from the Vertica nodes that are operating on it, degrading performance. This behavior can also occur when reading ORC or Parquet files if Vertica is not running on all Hadoop nodes. (If you are using separate Vertica and Hadoop clusters, all Hadoop access is over the network, and the performance cost is less noticeable.)

To prevent the undesired movement of data blocks across the HDFS cluster, consider excluding Vertica nodes from rebalancing. See the Hadoop documentation to learn how to do this.

Replication Factor

By default, HDFS stores three copies of each data block. Vertica is generally set up to store two copies of each data item through K-Safety. Thus, lowering the replication factor to 2 can save space and still provide data protection.

To lower the number of copies HDFS stores, set `HadoopFSReplication`, as explained in [Troubleshooting HDFS Storage Locations](#).

Disk Space for Non-HDFS Use

You also need to reserve some disk space for non-HDFS use. To reserve disk space using Ambari, set `dfs.datanode.du.reserved` to a value in the `hdfs-site.xml` configuration file.

Setting this parameter preserves space for non-HDFS files that Vertica requires.

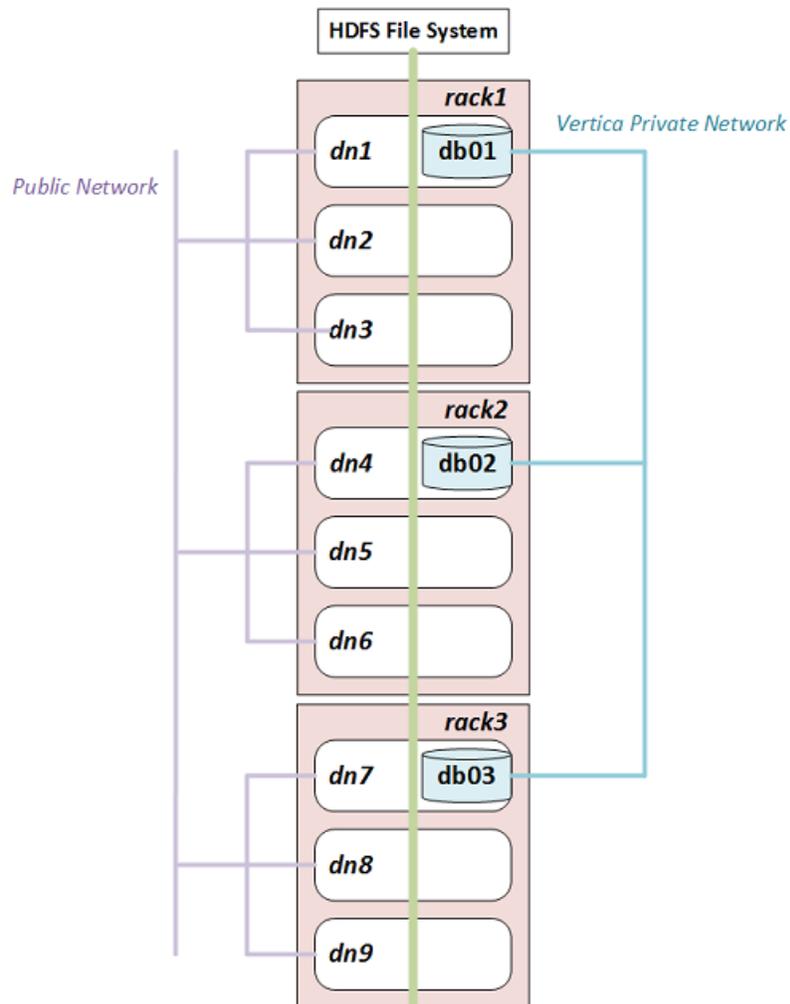
Configuring Rack Locality

Note: This feature is supported only for reading ORC and Parquet data on co-located clusters. It is only meaningful on Hadoop clusters that span multiple racks.

When possible, when planning a query Vertica automatically uses database nodes that are co-located with the HDFS nodes that contain the data. Moving query execution closer to the data reduces network latency and can improve performance. This behavior, called node locality, requires no additional configuration.

When Vertica is co-located on only a subset of HDFS nodes, sometimes there is no database node that is co-located with the data. However, performance is usually better if a query uses a database node in the same rack. If configured with information about Hadoop rack structure, Vertica attempts to use a database node in the same rack as the data to be queried.

For example, the following diagram illustrates a Hadoop cluster with three racks each containing three data nodes. (Typical production systems have more data nodes per rack.) In each rack, Vertica is co-located on one node.



If you configure rack locality, Vertica uses db01 to query data on dn1, dn2, or dn3, and uses db02 and db03 for data on rack2 and rack3 respectively. Because HDFS replicates data, any given data block can exist in more than one rack. If a data block is replicated on dn2, dn3, and dn6, for example, Vertica uses either db01 or db02 to query it.

Hadoop components are rack-aware, so configuration files describing rack structure already exist in the Hadoop cluster. To use this information in Vertica, configure fault groups that describe this rack structure. Vertica uses fault groups in query planning.

Configuring Fault Groups

Vertica uses [Fault Groups](#) to describe physical cluster layout. Because your database nodes are co-located on HDFS nodes, Vertica can use the information about the physical layout of the HDFS cluster.

Tip: For best results, ensure that each Hadoop rack contains at least one co-located Vertica node.

Hadoop stores its cluster-layout data in a topology mapping file in HADOOP_CONF_DIR. On HortonWorks the file is typically named topology_mappings.data. On Cloudera it is typically named topology.map. Use the data in this file to create an input file for the fault-group script. For more information about the format of this file, see [Creating a Fault Group Input File](#).

Following is an example topology mapping file for the cluster illustrated previously:

```
[network_topology]
dn1.example.com=/rack1
10.20.41.51=/rack1
dn2.example.com=/rack1
10.20.41.52=/rack1
dn3.example.com=/rack1
10.20.41.53=/rack1
dn4.example.com=/rack2
10.20.41.71=/rack2
dn5.example.com=/rack2
10.20.41.72=/rack2
dn6.example.com=/rack2
10.20.41.73=/rack2
dn7.example.com=/rack3
10.20.41.91=/rack3
dn8.example.com=/rack3
10.20.41.92=/rack3
dn9.example.com=/rack3
10.20.41.93=/rack3
```

From this data, you can create the following input file describing the Vertica subset of this cluster:

```
/rack1 /rack2 /rack3
/rack1 = db01
/rack2 = db02
/rack3 = db03
```

This input file tells Vertica that the database node "db01" is on rack1, "db02" is on rack2, and "db03" is on rack3. In creating this file, ignore Hadoop data nodes that are not also Vertica nodes.

After you create the input file, run the fault-group tool:

```
$ python /opt/vertica/scripts/fault_group_ddl_generator.py dbName input_file > fault_group_ddl.sql
```

The output of this script is a SQL file that creates the fault groups. Execute it following the instructions in [Creating Fault Groups Using the Fault Group Script](#).

You can review the new fault groups with the following statement:

```
=> SELECT member_name,node_address,parent_name FROM fault_groups  
INNER JOIN nodes ON member_name=node_name ORDER BY parent_name;
```

member_name	node_address	parent_name
db01	10.20.41.51	/rack1
db02	10.20.41.71	/rack2
db03	10.20.41.91	/rack3

(3 rows)

Working With Multi-Level Racks

A Hadoop cluster can use multi-level racks. For example, `/west/rack-w1`, `/west/rack-2`, and `/west/rack-w3` might be served from one data center, while `/east/rack-e1`, `/east/rack-e2`, and `/east/rack-e3` are served from another. Use the following format for entries in the input file for the fault-group script:

```
/west /east  
/west = /rack-w1 /rack-w2 /rack-w3  
/east = /rack-e1 /rack-e2 /rack-e3  
/rack-w1 = db01  
/rack-w2 = db02  
/rack-w3 = db03  
/rack-e1 = db04  
/rack-e2 = db05  
/rack-e3 = db06
```

Do not create entries using the full rack path, such as `/west/rack-w1`.

Auditing Results

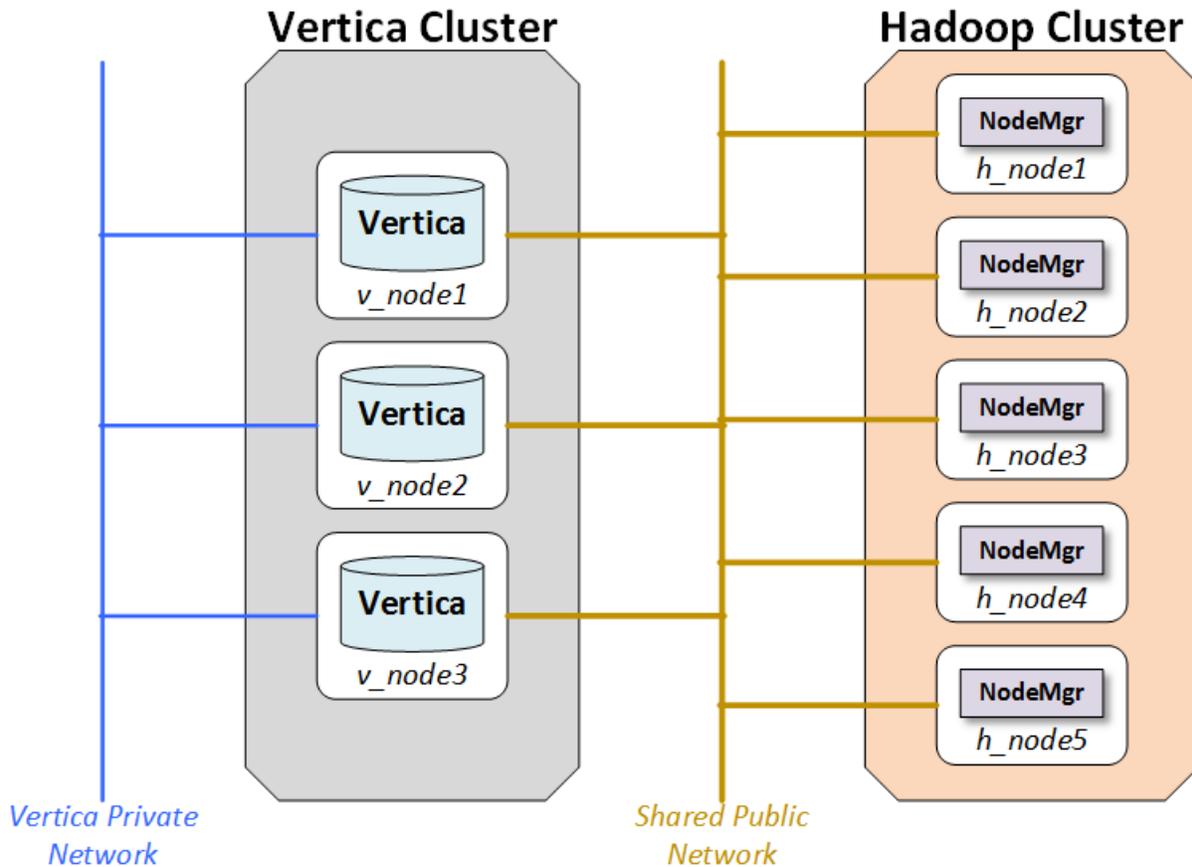
To see how much data can be loaded with rack locality, use [EXPLAIN](#) with the query and look for statements like the following in the output:

```
100% of ORC data including co-located data can be loaded with rack locality.
```

Separate Clusters

With separate clusters, a Vertica cluster and a Hadoop cluster share no nodes. You should use a high-bandwidth network connection between the two clusters.

The following figure illustrates the configuration for separate clusters::



Network

The network is a key performance component of any well-configured cluster. When Vertica stores data to HDFS it writes and reads data across the network.

The layout shown in the figure calls for two networks, and there are benefits to adding a third:

- **Database Private Network:** Vertica uses a private network for command and control and moving data between nodes in support of its database functions. In some networks, the command and control and passing of data are split across two networks.
- **Database/Hadoop Shared Network:** Each Vertica node must be able to connect to each Hadoop data node and the Name Node. Hadoop best practices generally require a dedicated network for the Hadoop cluster. This is not a technical requirement, but a dedicated network improves Hadoop performance. Vertica and Hadoop should share the dedicated Hadoop network.

- **Optional Client Network:** Outside clients may access the clustered networks through a client network. This is not an absolute requirement, but the use of a third network that supports client connections to either Vertica or Hadoop can improve performance. If the configuration does not support a client network, than client connections should use the shared network.

Hadoop Configuration Parameters

For best performance, set the following parameters with the specified minimum values:

Parameter	Minimum Value
HDFS block size	512MB
Namenode Java Heap	1GB
Datanode Java Heap	2GB

Hadoop Interfaces

Vertica provides several ways to interact with data stored in HDFS, described below. Decisions about [Cluster Layout](#) can affect the decisions you make about which Hadoop interfaces to use.

Querying Data Stored in HDFS

Vertica can query data directly from HDFS without requiring you to copy data. To query data, you define an external table as for any other external data source. (This option does not make use of schema definitions from Hive.)

Vertica can query the data directly from the HDFS nodes, bypassing the WebHDFS service. To query data directly, Vertica needs access to HDFS configuration files.

See [Reading Directly from HDFS](#) and [Reading Hadoop Columnar File Formats](#).

Querying Data Using the HCatalog Connector

The HCatalog Connector uses Hadoop services (Hive and HCatalog) to query data stored in HDFS. Using the HCatalog Connector, Vertica can use Hive's schema definitions. However, performance can be poor compared to defining your own external table and querying the data directly. The HCatalog Connector is also sensitive to changes in the Hadoop libraries on which it depends; upgrading your Hadoop cluster might affect your HCatalog connections.

See [Using the HCatalog Connector](#).

Using ROS Data

Storing data in the Vertica native file format (ROS) delivers better query performance than reading externally-stored data. You can create storage locations on your HDFS cluster to hold ROS data. Even if your data is already stored in HDFS, you might choose to copy that data into an HDFS storage location for better performance. If your Vertica cluster also uses local file storage, you can use HDFS storage locations for lower-priority data.

See [Using HDFS Storage Locations](#) for information about creating and managing HDFS storage locations, and [Reading Directly from HDFS](#) for information about copying data into them.

Exporting Data

You might want to export data from Vertica, either to share it with other Hadoop-based applications or to move lower-priority data from ROS to less-expensive storage. You can export a table, or part of one, in Hadoop columnar format. After export you can still query the data using an external table, as for any other data.

See [Exporting Data](#).

Using Kerberos with Hadoop

If your Hortonworks or Cloudera Hadoop cluster uses Kerberos authentication to restrict access to HDFS, you must configure Vertica to make authenticated connections. The details of this configuration vary, based on which methods you are using to access HDFS data.

Vertica does not support Kerberos for MapR.

This section covers the following topics:

- [How Vertica uses Kerberos With Hadoop](#)
- [Configuring Kerberos](#)

How Vertica uses Kerberos With Hadoop

Vertica authenticates with Hadoop in two ways that require different configurations:

- **User Authentication**—On behalf of the user, by passing along the user's existing Kerberos credentials, as occurs with the HDFS Connector and the HCatalog Connector.
- **Vertica Authentication**—On behalf of system processes (such as the Tuple Mover), by using a special Kerberos credential stored in a keytab file.

Note: Vertica and Hadoop must use the same Kerberos server (KDC).

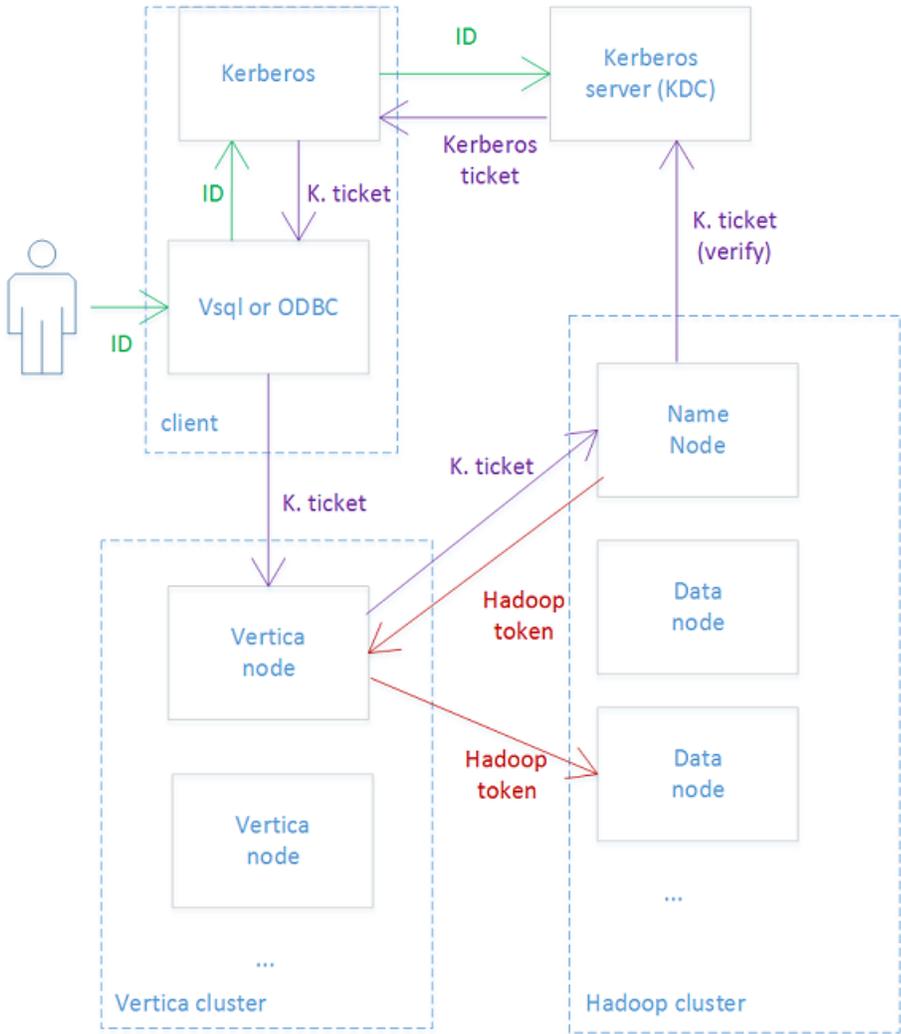
User Authentication

To use Vertica with Kerberos and Hadoop, the client user first authenticates with the Kerberos server (Key Distribution Center, or KDC) being used by the Hadoop cluster. A user might run `kinit` or sign in to Active Directory, for example.

A user who authenticates to a Kerberos server receives a Kerberos ticket. At the beginning of a client session, Vertica automatically retrieves this ticket. The database then uses this ticket to get a Hadoop token, which Hadoop uses to grant access. Vertica uses this token to access HDFS, such as when executing a query on behalf of the user. When the token expires, the database automatically renews it, also renewing the Kerberos ticket if necessary.

The user must have been granted permission to access the relevant files in HDFS. This permission is checked the first time Vertica reads HDFS data.

The following figure shows how the user, Vertica, Hadoop, and Kerberos interact in user authentication:

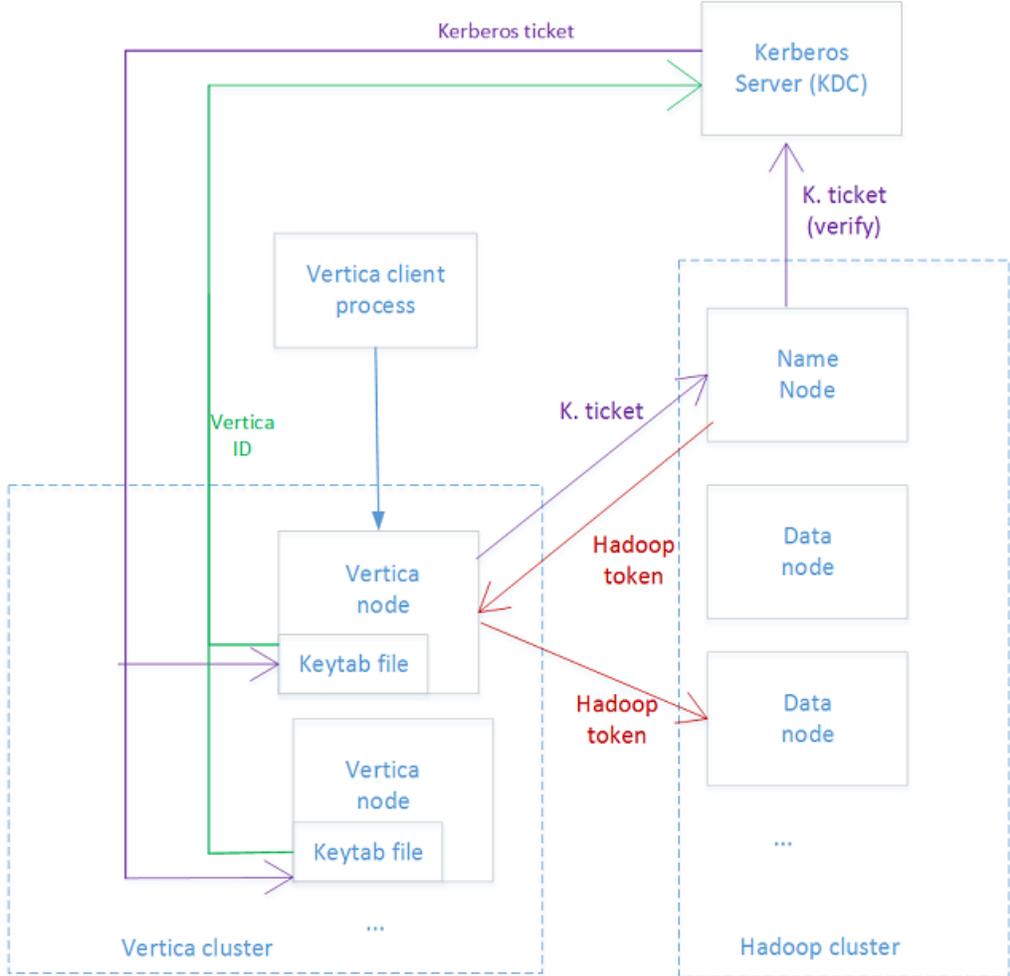


When using the HDFS Connector or the HCatalog Connector, or when reading an ORC or Parquet file stored in HDFS, Vertica uses the client identity as the preceding figure shows.

Vertica Authentication

Automatic processes, such as the Tuple Mover, do not log in the way users do. Instead, Vertica uses a special identity (principal) stored in a keytab file on every database node. (This approach is also used for Vertica clusters that use Kerberos but do not use Hadoop.) After you configure the keytab file, Vertica uses the principal residing there to automatically obtain and maintain a Kerberos ticket, much as in the client scenario. In this case, the client does not interact with Kerberos.

The following figure shows the interactions required for Vertica authentication:



Each Vertica node uses its own principal; it is common to incorporate the name of the node into the principal name. You can either create one keytab per node, containing only that node's principal, or you can create a single keytab containing all the principals and distribute the file to all nodes. Either way, the node uses its principal to get a Kerberos ticket and then uses that ticket to get a Hadoop token. For simplicity, the preceding figure shows the full set of interactions for only one database node.

When creating HDFS storage locations Vertica uses the principal in the keytab file, not the principal of the user issuing the CREATE LOCATION statement.

See Also

For specific configuration instructions, see [Configuring Kerberos](#).

Configuring Kerberos

Vertica can connect with Hadoop in several ways, and how you manage Kerberos authentication varies by connection type. If you use Kerberos, you must use it for both your HDFS and Vertica clusters.

Prerequisite: Set Up Users and the Keytab File

If you have not already configured Kerberos authentication for Vertica, follow the instructions in [Configure Vertica for Kerberos Authentication](#). Of particular importance for Hadoop integration:

1. Create one Kerberos principal per node.
2. Place the keytab file(s) in the same location on each database node and set `KerberosKeytabFile` to its location. (See [Specify the Location of the Keytab File](#).)
3. Set `KerberosServiceName` to the name of the principal. (See [Inform About the Kerberos Principal](#).)

Reads with the hdfs Scheme

Vertica can access files stored in HDFS using the `hdfs` URL scheme instead of using `WebHDFS`. Vertica authenticates using the current user's Kerberos principal, not the database's Kerberos principal. No additional Kerberos-specific configuration is required.

HCatalog Connector

You use the HCatalog Connector to query data in Hive. The HCatalog Connector executes queries on behalf of Vertica users. If the current user has a Kerberos key, then Vertica passes that key to the HCatalog connector automatically. Verify that the administrator of your HDFS cluster has granted HDFS access to all users who need access to Hive.

In addition, in your Hadoop configuration files (`core-site.xml` in most distributions), make sure that you enable all Hadoop components to impersonate the Vertica user. The easiest way to do so is to set the `proxyuser` property using wildcards for all users on all hosts and in all groups.

Consult your Hadoop documentation for instructions. Make sure you set this property before running `hcatUtil` (see [Configuring Vertica for HCatalog](#)).

HDFS Storage Location

You can create a database storage location in HDFS. An HDFS storage location provides improved performance compared to other HDFS interfaces (such as the HCatalog Connector). By storing the data in Vertica, rather than creating an external table, you can reduce query response times.

To use a storage location in HDFS with Kerberos, take the following steps:

1. Create a Kerberos principal for each Vertica node as described under [Prerequisites](#).
2. Give all node principals read and write permission to the HDFS directory you will use as a storage location.

If you plan to back up your HDFS storage locations, take the following additional steps:

1. Grant Hadoop superuser privileges to the new principals.
2. Configure backups, including setting the `HadoopConfigDir` configuration parameter, following the instructions in [Configuring Hadoop and Vertica to Enable Backup of HDFS Storage](#).
3. Configure user impersonation to be able to restore from backups following the instructions in "Setting Kerberos Parameters" in [Configuring Vertica to Restore HDFS Storage Locations](#).

Because the keytab file supplies the principal used to create the location, you must have it in place before creating the storage location. After you deploy keytab files to all database nodes, use the `CREATE LOCATION` statement to create the storage location as usual.

HDFS Connector

The HDFS Connector loads data from HDFS into Vertica on behalf of the user. If the user performing the data load has a Kerberos key, then the connector uses it to access HDFS. Verify that all users who use this connector have been granted access to HDFS.

The HDFS Connector is deprecated. Use the other Hadoop interfaces instead.

Verifying Kerberos Configuration

Use the [KERBEROS_HDFS_CONFIG_CHECK](#) metafunction to verify that Vertica can use Kerberos to access HDFS. You can call it with no parameters to test all paths described in the Hadoop configuration files. Alternatively, you can specify `hdfs`, `webhdfs`, and `WebHCat` servers to test individually.

```
=> SELECT KERBEROS_HDFS_CONFIG_CHECK();

=> SELECT KERBEROS_HDFS_CONFIG_CHECK('node1.example.com:9433',
                                     'node2.example.com:10443', 'node2.example.com:14443');
```

This function does not yet check access to HiveServer2.

Token Expiration

Vertica attempts to automatically refresh Hadoop tokens before they expire, but you can also set a minimum refresh frequency if you prefer. Use the `HadoopFSTokenRefreshFrequency` configuration parameter to specify the frequency in seconds:

```
=> ALTER DATABASE exampledb SET HadoopFSTokenRefreshFrequency = '86400';
```

If the current age of the token is greater than the value specified in this parameter, Vertica refreshes the token before accessing data stored in HDFS.

See Also

- [How Vertica uses Kerberos With Hadoop](#)
- [Troubleshooting Kerberos Authentication](#)

Reading Directly from HDFS

When reading files from HDFS, you can use the `hdfs` scheme instead of the `webhdfs` scheme. Using the `hdfs` scheme can improve performance and stability by bypassing the WebHDFS service.

You can use the `hdfs` scheme with [COPY](#) and with [CREATE EXTERNAL TABLE AS COPY](#). When using the `hdfs` scheme with `COPY`, you do not need to specify `ON ANY NODE`.

To support direct access, Vertica requires access to certain configuration files from your HDFS cluster. See [Configuring the hdfs Scheme](#).

URL Format

You specify the location of a file in HDFS using a URL. In most cases, you use the `hdfs:///` URL prefix (three slashes), and then specify the file path. Vertica uses the `fs.defaultFS` Hadoop configuration parameter to find the data. The following example loads data stored in HDFS.

```
=> COPY t FROM 'hdfs:///opt/data/file1.dat';
```

You can specify a host and port explicitly using the following format: `hdfs://host:port/`. The specified host is the Name Node, not an individual data node. If you are using High Availability (HA) Name Nodes you should not use an explicit host because high availability is provided through nameservices instead.

Your HDFS cluster might use High Availability Name Nodes or define nameservices. If so, you should use the nameservice instead of the host and port, in the format `hdfs://nameservice/`. The nameservice you specify must be defined in `hdfs-site.xml`.

The following example shows how you can use a nameservice, `hadoopNS`, with the `hdfs` scheme.

```
=> CREATE EXTERNAL TABLE tt (a1 INT, a2 VARCHAR(20))
    AS COPY FROM 'hdfs://hadoopNS/data/file.csv';
```

If you are using Vertica to access data from more than one HDFS cluster, always use explicit nameservices or hosts in the URL. Using `hdfs:///` could produce unexpected results because Vertica uses the first value of `fs.defaultFS` that it finds. To access multiple HDFS clusters, you must use host and service names that are globally unique. See [Configuring the hdfs Scheme](#) for more information.

Note: All characters in URLs that are not a–z, A–Z, 0–9, '-', '.', '_' or '~' must be converted to URL encoding (%NN where NN is a two-digit hexadecimal number). For example, use %20 for space.

Kerberos Authentication

If the file you want to read resides on an HDFS cluster that uses Kerberos authentication, Vertica uses the current user's principal to authenticate. It does *not* use the database's principal.

You can use the [KERBEROS_HDFS_CONFIG_CHECK](#) metafunction to verify that Vertica is correctly configured for Kerberos access.

Configuring the hdfs Scheme

Vertica uses information from the Hadoop cluster configuration to support the hdfs scheme. Vertica nodes therefore must have access to certain Hadoop configuration files. To use a cluster with High Availability Name Node or to read from more than one Hadoop cluster, you must perform additional configuration.

For both co-located and separate clusters that use Kerberos authentication, configure Vertica for Kerberos as explained in [Configure Vertica for Kerberos Authentication](#).

Using the hdfs scheme still requires access to the WebHDFS service. For some special cases, Vertica cannot use the hdfs scheme and falls back to webhdfs.

Accessing Hadoop Configuration Files

To support the hdfs scheme, your Vertica nodes need access to certain Hadoop configuration files:

- If Vertica is co-located on HDFS nodes, then those configuration files are already present.
- If Vertica is running on a separate cluster, you must copy the required files to all database nodes. A simple way to do so is to configure your Vertica nodes as Hadoop edge nodes. Client applications run on *edge nodes*; from Hadoop's perspective, Vertica is a client application. You can use Ambari or Cloudera Manager to configure edge nodes. For more information, see the documentation from your Hadoop vendor.

Verify that the value of the HadoopConfDir configuration parameter (see [Hadoop Parameters](#)) includes a directory containing the files listed in the following table. If you do not set a value, Vertica looks for the files in /etc/hadoop/conf. You can use the [VERIFY_HADOOP_CONF_DIR](#) meta-function to verify that Vertica can find configuration files.

Vertica uses the following configuration files and properties. If a property is not defined, Vertica uses the defaults shown in the table. Your Hadoop configuration files must specify all properties that have no defaults.

File	Properties	Default
core-site.xml	fs.defaultFS	none
hdfs-site.xml	dfs.client.failover.max.attempts	4
	dfs.client.failover.connection.retries.on.timeouts	0
	ipc.client.connect.timeout	30 seconds
	ipc.client.connect.retry.interval	10 seconds
	(For HA NN:) dfs.nameservices	none

Using a Cluster with High Availability Name Nodes

If your Hadoop cluster uses High Availability (HA) Name Nodes, verify that the dfs.nameservices parameter and the individual name nodes are defined in hdfs-site.xml.

Using More Than One Hadoop Cluster

In some cases, a Vertica cluster requires access to more than one HDFS cluster. For example, your business might use separate HDFS clusters for separate regions, or you might need data from both test and deployment clusters.

To support multiple clusters, perform the following steps:

1. Copy the configuration files from all HDFS clusters to your database nodes. You can place the copied files in any location readable by Vertica. However, as a best practice, you should place them all in the same directory tree, with one subdirectory per HDFS cluster. The locations must be the same on all database nodes.

2. Set the `HadoopConfDir` configuration parameter. The value is a colon-separated path containing the directories for all of your HDFS clusters.
3. Use an explicit name node or name service in the URL when creating an external table or copying data. Do not use `hdfs://` because it could be ambiguous. For more information about URLs, see [URL Format](#).

Vertica connects directly to a name node or name service; it does not otherwise distinguish among HDFS clusters. Therefore, names of HDFS name nodes and name services must be globally unique.

Updating Configuration Files

If you update the configuration files after starting Vertica, use the following statement to refresh them:

```
=> SELECT CLEAR_HDFS_CACHES();
```

The [CLEAR_HDFS_CACHES](#) function also flushes information about which Name Node is active in a High Availability (HA) Hadoop cluster. Therefore, the first `hdfs` request after calling this function is slow, because the initial connection to the Name Node can take more than 15 seconds.

Troubleshooting Reads from the `hdfs` Scheme

You might encounter the following issues when using the `hdfs` URL scheme to access data in HDFS.

WebHDFS Error When Using `hdfs` URLs

When creating an external table or loading data and using the `hdfs` scheme, you might see errors from WebHDFS failures. Such errors indicate that Vertica was not able to use the `hdfs` scheme and fell back to `webhdfs`, but that the WebHDFS configuration is incorrect. Verify that the HDFS configuration files in `HadoopConfDir` have the correct WebHDFS configuration for your Hadoop cluster. See [Configuring the `hdfs` Scheme](#) for information about use of these files. See your Hadoop documentation for information about WebHDFS configuration.

Queries Take a Long Time to Run When Using HA

The High Availability Name Node feature in HDFS allows a name node to fail over to a standby name node. The `dfs.client.failover.max.attempts` configuration parameter (in `hdfs-site.xml`) specifies how many attempts to make when failing over. Vertica uses a default value of 4 if this parameter is not set. After reaching the maximum number of failover attempts, Vertica concludes that the HDFS cluster is unavailable and aborts the operation. A second parameter, `ipc.client.connect.retry.interval`, specifies the time to wait between attempts, with typical values being 10 to 20 seconds.

Cloudera and Hortonworks both provide tools to automatically generate configuration files. These tools can set the maximum number of failover attempts to a much higher number (50 or 100). If the HDFS cluster is unavailable (all name nodes are unreachable), Vertica can appear to hang for an extended period (minutes to hours) while trying to connect.

Failover attempts are logged in the [QUERY_EVENTS](#) system table. The following example shows how to query this table to find these events:

```
=> SELECT event_category, event_type, event_description, operator_name,
event_details, count(event_type) AS count
FROM query_events
WHERE event_type ilike 'LibHDFS++ FAILOVER RETRY'
GROUP BY event_category, event_type, event_description, operator_name, event_details;
-[ RECORD 1 ]-----+-----
event_category      | EXECUTION
event_type          | LibHDFS++ FAILOVER RETRY
event_description   | LibHDFS++ Namenode failover and retry.
operator_name       | LibHDFS++ FileSystem
event_details       | Libhdfs++ request failed on ns
count               | 4
```

You can either wait for Vertica to complete or abort the connection, or set the `dfs.client.failover.max.attempts` parameter to a lower value.

Reading Hadoop Columnar File Formats

For data in certain Hadoop columnar formats, you can create external tables or copy data into Vertica directly, without going through Hive to get the metadata. Currently, Vertica supports direct reading for the ORC (Optimized Row Columnar) and Parquet formats. Vertica can read ORC and Parquet data from a local file system or from HDFS.

Vertica can also export data in the Parquet format. See [Exporting Data](#).

Requirements

ORC or Parquet files must not use complex data types. Vertica supports all simple data types supported in Hive version 0.11 or later.

Files compressed by Hive or Impala require Zlib (GZIP) or Snappy compression. Vertica does not support LZO compression for these formats.

Creating External Tables

In the [CREATE EXTERNAL TABLE AS COPY](#) statement, specify a format of ORC or PARQUET as follows:

```
=> CREATE EXTERNAL TABLE tableName (columns)
    AS COPY FROM path ORC[(hive_partition_cols=partitions)];
=> CREATE EXTERNAL TABLE tableName (columns)
    AS COPY FROM path PARQUET[(hive_partition_cols=partitions)];
```

Set the value of *path* based on where the file is located. If the file resides:

- On the local file system of the node where you issue the command—Use a local file path. Escape special characters in file paths with backslashes.
- In HDFS—Use the `hdfs` scheme and then append the file path. Escape special characters in HDFS paths using URL encoding. HDFS URLs usually begin with `hdfs://`. See [Reading Directly from HDFS](#) for more information about URL format.

When defining an external table, you must define all of the data columns in the file. You may omit partition columns. Unlike with some other data sources, you cannot select only the data columns of interest. If you omit data columns, the ORC or Parquet reader aborts with an error.

If the data is partitioned you must alter the path value and specify the `hive_partition_cols` argument for the ORC or PARQUET parameter. You must also list partitioned columns last in `columns`. See [Using Partition Columns](#).

If `path` is a path on the local file system on a Vertica node, specify the node using ON NODE in the COPY statement. Do not use COPY LOCAL. If `path` is in HDFS, COPY defaults to ON ANY NODE so you do not need to specify it.

Files stored in HDFS are governed by HDFS privileges. For files stored on the local disk, however, Vertica requires that users be granted access. All users who have administrative privileges have access. For other users, you must create a storage location and grant access to it. See [CREATE EXTERNAL TABLE AS COPY](#). Only users who have access to both the Vertica user storage location and the HDFS directory can read from the table.

Loading Data

In the `COPY` statement, specify a format of ORC or PARQUET:

```
=> COPY tableName FROM path ORC[(hive_partition_cols='partitions')];  
=> COPY tableName FROM path PARQUET[(hive_partition_cols='partitions')];
```

As with external tables, `path` may be a local or `hdfs` : `path`. For information about the `hive_partition_cols` parameter, see [Using Partition Columns](#).

Be aware that if you load from multiple files in the same COPY statement, and any of them is aborted, the entire load aborts. This behavior differs from that for delimited files, where the COPY statement loads what it can and ignores the rest.

Supported Data Types

Vertica can natively read columns of all data types supported in Hive version 0.11 and later except for complex types. If the data contains complex types such as maps, the COPY or CREATE EXTERNAL TABLE AS COPY statement aborts with an error message. Vertica does not attempt to read only some columns; either the entire file is read or the operation fails. For a complete list of supported types, see [HIVE Data Types](#).

Examples

The following example shows how you can read from all ORC files in a local directory. This example uses all supported data types.

```
=> CREATE EXTERNAL TABLE t (a1 TINYINT, a2 SMALLINT, a3 INT, a4 BIGINT, a5 FLOAT,  
                             a6 DOUBLE PRECISION, a7 BOOLEAN, a8 DATE, a9 TIMESTAMP,  
                             a10 VARCHAR(20), a11 VARCHAR(20), a12 CHAR(20), a13 BINARY(20),  
                             a14 DECIMAL(10,5))  
    AS COPY FROM '/data/orc_test_*.orc' ORC;
```

The following example shows how to use a name service with the `hdfs` scheme. This example assumes that the name service, `hadoopNS`, is defined in the Hadoop configuration files that were copied to the Vertica cluster.

```
=> CREATE EXTERNAL TABLE tt (a1 INT, a2 VARCHAR(20))  
    AS COPY FROM 'hdfs://hadoopNS/data/file.parquet' PARQUET;
```

Using Partition Columns

An ORC or Parquet file contains data columns. To these files you can add partition columns at write time. The data files do not store values for partition columns; instead, when writing the files you divide them into groups (partitions) based on column values. You can use partitioning to improve the performance of queries that restrict results by the partitioned column.

For example, if you have a table with a date column, and you know you will be writing queries restricted to particular dates, you can partition by date. Thus, Vertica can skip reading some files entirely when executing your date-restricted queries. This behavior is called partition pruning.

You can create partitions regardless of where you store the files—in HDFS, on a local file system, or in a shared file system such as NFS.

You can use Hive or the Vertica Parquet Writer to create partitions, or you can create them manually. See [Partitioning Hive Tables](#) for information about tuning partitions.

Partition Structure

By default, both Hive and Vertica write Hadoop columnar format files that contain the data for all table columns without partitioning. The column data is laid out in stripes, or groups of row

data. When Vertica loads this data it reads all of the stripes.

If you partition the data, however, you can avoid writing some of that data into the files and thus reduce the amount to be read. Instead of storing a column's data in the files, you create a directory structure that partitions the data based on the value in a column.

For example, if the data includes a date column, you can write each date as a separate partition. Each partition is a directory with a name of the form "column=value". If you have a date column named "created" that is partitioned by day, you would have the following directory structure:

```
path/created=2016-11-01/*
path/created=2016-11-02/*
path/created=2016-11-03/*
path/...
```

As this example shows, the files in each subdirectory contain all columns *except* the "created" column.

You can partition by more than one column, creating a layered structure. For example, adding another partitioned column, "region", to the preceding example would produce the following directory structure:

```
path/created=2016-11-01/region=northeast/*
path/created=2016-11-01/region=central/*
path/created=2016-11-01/region=southeast/*
path/created=2016-11-01/...
path/created=2016-11-02/region=northeast/*
path/created=2016-11-02/region=central/*
path/created=2016-11-02/region=southeast/*
path/created=2016-11-02/...
path/created=2016-11-03/...
path/...
```

With this change, the data files contain all columns except "created" and "region".

Note: The files must contain at least one real (not partitioned) column. You cannot partition by every column in a table.

You can create partitions for columns of any simple data type. As a best practice, however, you should avoid partitioning columns with BOOLEAN, FLOAT, and NUMERIC types. Vertica does not prune partitions of these types, so they do not provide a performance advantage.

Under some circumstances Hive writes a partition with a value of `__HIVE_DEFAULT_PARTITION__`. Vertica treats these values as NULL.

COPY Syntax

When creating an external table from partitioned data, you must do all of the following:

- In the column definition in the table, list the partition columns last and in order.
- In the path, use wildcards to include all of the levels of directories and files.
- In the ORC or PARQUET statement, specify the partition columns, in order, in the `hive_partition_cols` parameter. (The argument name is the same even if you didn't use Hive to do the partitioning; it refers to Hive-style partitions.)

The following example creates an external table using the partitioned data shown previously. The table includes four columns. Two columns, "id" and "name", are in the data files. The other two, "created" and "region", are partitioned.

```
=> CREATE EXTERNAL TABLE t (id int, name varchar(50), created date, region varchar(50))
  AS COPY FROM 'hdfs:///path/**/*'
  ORC(hive_partition_cols='created,region');
```

The path includes one wildcard (*) for each level of directory partitioning and then one more for the files. The number of wildcards must always be one more than the number of partitioned columns.

You do not need to include all of the partitioned columns in `hive_partition_cols` if those columns are not relevant for your queries. However, the partition columns must be the last columns in the table definition. For example, you can define the following table for the partitioned data shown previously:

```
=> CREATE EXTERNAL TABLE t2 (id int, name varchar(50), created date, region varchar(50))
  AS COPY FROM 'hdfs:///path/**/*' ORC(hive_partition_cols='region');
```

Values in the "created" column are all null because no data appears in the files for that column and `hive_partition_cols` does not include it.

However, the following example produces an error.

```
=> CREATE EXTERNAL TABLE t3 (id int, name varchar(50), created date, region varchar(50))
  AS COPY FROM 'hdfs:///path/**/*' ORC(hive_partition_cols='created');
```

In this example, the table definition includes the "region" column after the "created" column, and "region" is not included in `hive_partition_cols`. Because this column is not listed as a partition column, Vertica interprets it as a data column and produces an error because the column is not present.

If Vertica cannot convert a partition value to the declared type for that column, it sets the value to NULL. The following example incorrectly declares region to be an integer rather than a varchar.

```
=> CREATE EXTERNAL TABLE t4 (id int, name varchar(50), created date, region int)
AS COPY FROM 'hdfs:///path/**/*/' ORC(hive_partition_cols='region');
```

Vertica cannot coerce a directory named "region=northeast" into an integer value, so it sets that column value to NULL for all rows it reads from this directory. If you declare the column with IS NOT NULL, Vertica rejects the row. If the number of rows exceeds REJECTMAX, Vertica reports an error.

Note: If you change how files are partitioned on disk, you must re-create your external tables.

Queries

When executing queries with predicates, Vertica skips subdirectories that do not satisfy the predicate. This process is called *partition pruning* and it can significantly improve query performance. See [Improving Query Performance for Data Stored in HDFS](#) for more information about partition pruning and other techniques for optimizing queries.

The following example reads only the partitions for the specified region, for all dates. Although the data is also partitioned by date, the query does not restrict the date.

```
=> SELECT * FROM t WHERE region='northeast';
```

To verify that Vertica is pruning partitions, look in the explain plan for a message similar to the following:

```
files with unmatched Hive partition have been pruned
```

Improving Query Performance for Data Stored in HDFS

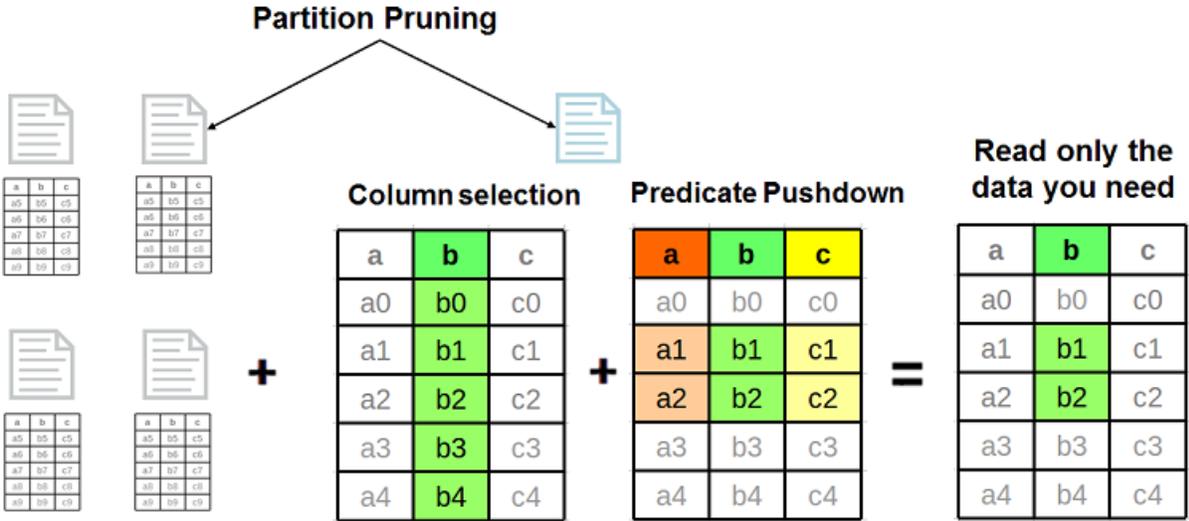
When working with external tables in Hadoop columnar formats, Vertica tries to improve performance in the following ways:

- By pushing query execution closer to the data so less has to be read and transmitted. Vertica uses the following specific techniques: predicate pushdown, column selection, and

partition pruning.

- By taking advantage of data locality in the query plan.
- By analyzing the row count to get the best join orders in the query plan.

The following figure illustrates optimizations that can reduce the amount of data to be read:



Tuning ORC Stripes and Parquet Rowgroups

Vertica can read ORC and Parquet files generated by any Hive version. However, newer Hive versions store more metadata in these files. This metadata is used by both Hive and Vertica to prune values and to read only the required data. Use the latest Hive version to store data in these formats. ORC and Parquet are fully forward- and backward-compatible. To get the best performance, use Hive 0.14 or later.

The ORC format splits a table into groups of rows called stripes and stores column-level metadata in each stripe. The Parquet format splits a table into groups of rows called rowgroups and stores column-level metadata in each rowgroup. Each stripe/rowgroup's metadata is used during predicate evaluation to determine whether the values from this stripe need to be read. Large stripes usually yield better performance, so set the stripe size to at least 256M.

Hive writes ORC stripes and Parquet rowgroups to HDFS, which stores data in HDFS blocks distributed among multiple physical data nodes. Accessing an HDFS block requires opening a separate connection to the corresponding data node. It is advantageous to ensure that an ORC stripe or Parquet rowgroup does not span more than one HDFS block. To do so, set the HDFS block size to be larger than the stripe/rowgroup size. Setting HDFS block size to 512M is usually sufficient.

Hive provides three compression options: None, Snappy, and Zlib. Use Snappy or Zlib compression to reduce storage and I/O consumption. Usually, Snappy is less CPU-intensive but can yield lower compression ratios compared to Zlib.

Storing data in sorted order can improve data access and predicate evaluation performance. Sort table columns based on the likelihood of their occurrence in query predicates; columns that most frequently occur in comparison or range predicates should be sorted first.

Partitioning tables is a very useful technique for data organization. Similarly to sorting tables by columns, partitioning can improve data access and predicate evaluation performance. Vertica supports Hive-style partitions and partition pruning.

The following Hive statement creates an ORC table with stripe size 256M and Zlib compression:

```
hive> CREATE TABLE customer_visits (  
    customer_id bigint,  
    visit_num int,  
    page_view_dt date)  
    STORED AS ORC tblproperties("orc.compress"="ZLIB",  
    "orc.stripe.size"="268435456");
```

The following statement creates a Parquet table with stripe size 256M and Zlib compression:

```
hive> CREATE TABLE customer_visits (  
    customer_id bigint,  
    visit_num int,  
    page_view_dt date)  
    STORED AS PARQUET tblproperties("parquet.compression"="ZLIB",  
    "parquet.stripe.size"="268435456");
```

Predicate Pushdown and Column Selection

Predicate pushdown moves parts of the query execution closer to the data, reducing the amount of data that must be read from disk or across the network. ORC files have three levels of indexing: file statistics, stripe statistics, and row group indexes. Predicates are applied only to the first two levels. Parquet files have two levels of statistics: rowgroup statistics and page statistics. Predicates are only applied to the first level.

Predicate pushdown is automatically applied for files written with Hive version 0.14 and later. ORC files written with earlier versions of Hive might not contain the required statistics. When executing a query against a file that lacks these statistics, Vertica logs an `EXTERNAL_PREDICATE_PUSHDOWN_NOT_SUPPORTED` event in the [QUERY_EVENTS](#) system table. If you are seeing performance problems with your queries, check this table for these events.

Another query performance optimization technique used by Vertica is *column selection*. Vertica reads from ORC or Parquet files only the columns specified in the query statement. For

example, the following statement reads only the `customer_id` and `visit_num` columns from the corresponding ORC files:

```
=> CREATE EXTERNAL TABLE customer_visits (  
    customer_id bigint,  
    visit_num int,  
    page_view_dt date)  
    AS COPY FROM '...' ORC;  
  
=> SELECT customer_id from customer_visits  
    WHERE visit_num > 10;
```

Data Locality

In a cluster where Vertica nodes are co-located on HDFS nodes, the query can use data locality to improve performance. For Vertica to do so, *both* the following conditions must exist::

- The data is on an HDFS node where a database node is also present.
- The query is not restricted to specific nodes using `ON NODE`.

When both these conditions exist, the query planner uses the co-located database node to read that data locally, instead of making a network call.

You can see how much data is being read locally by inspecting the query plan. The label for `LoadStep(s)` in the plan contains a statement of the form: "X% of ORC/Parquet data matched with co-located Vertica nodes". To increase the volume of local reads, consider adding more database nodes. HDFS data, by its nature, can't be moved to specific nodes, but if you run more database nodes you increase the likelihood that a database node is local to one of the copies of the data.

Creating Sorted Files in Hive

Unlike Vertica, Hive does not store table columns in separate files and does not create multiple projections per table with different sort orders. For efficient data access and predicate pushdown, sort Hive table columns based on the likelihood of their occurrence in query predicates. Columns that most frequently occur in comparison or range predicates should be sorted first.

Data can be inserted into Hive tables in a sorted order by using the `ORDER BY` or `SORT BY` keywords. For example, to insert data into the ORC table "customer_visit" from another table "visits" with the same columns, use these keywords with the `INSERT INTO` command:

```
hive> INSERT INTO TABLE customer_visits
      SELECT * from visits
      ORDER BY page_view_dt;
```

```
hive> INSERT INTO TABLE customer_visits
      SELECT * from visits
      SORT BY page_view_dt;
```

The difference between the two keywords is that ORDER BY guarantees global ordering on the entire table by using a single MapReduce reducer to populate the table. SORT BY uses multiple reducers, which can cause ORC or Parquet files to be sorted by the specified column(s) but not be globally sorted. Using the latter keyword can increase the time taken to load the file.

You can combine clustering and sorting to sort a table globally. The following table definition adds a hint that data is inserted into this table bucketed by customer_id and sorted by page_view_dt:

```
hive> CREATE TABLE customer_visits_bucketed (
      customer_id bigint,
      visit_num int,
      page_view_dt date)
      CLUSTERED BY (page_view_dt)
      SORTED BY (page_view_dt) INTO 10 BUCKETS
      STORED AS ORC;
```

When inserting data into the table, you must explicitly specify the clustering and sort columns, as in the following example:

```
hive> INSERT INTO TABLE customer_visits_bucketed
      SELECT * from visits
      DISTRIBUTE BY page_view_dt
      SORT BY page_view_dt;
```

The following statement is equivalent:

```
hive> INSERT INTO TABLE customer_visits_bucketed
      SELECT * from visits
      CLUSTER BY page_view_dt;
```

Both of the above commands insert data into the customer_visits_bucketed table, globally sorted on the page_view_dt column.

Partitioning Hive Tables

Table partitioning in Hive is an effective technique for data separation and organization, as well as for reducing storage requirements. To partition a table in Hive, include it in the PARTITIONED BY clause:

```
hive> CREATE TABLE customer_visits (  
        customer_id bigint,  
        visit_num int)  
    PARTITIONED BY (page_view_dt date)  
    STORED AS ORC;
```

Hive does not materialize partition column(s). Instead, it creates subdirectories of the following form:

```
path_to_table/partition_column_name=value/
```

When the table is queried, Hive parses the subdirectories' names to materialize the values in the partition columns. The value materialization in Hive is a plain conversion from a string to the appropriate data type.

Inserting data into a partitioned table requires specifying the value(s) of the partition column (s). The following example creates two partition subdirectories, "customer_visits/page_view_dt=2016-02-01" and "customer_visits/page_view_dt=2016-02-02":

```
hive> INSERT INTO TABLE customer_visits  
    PARTITION (page_view_dt='2016-02-01')  
    SELECT customer_id, visit_num from visits  
    WHERE page_view_dt='2016-02-01'  
    ORDER BY page_view_dt;  
  
hive> INSERT INTO TABLE customer_visits  
    PARTITION (page_view_dt='2016-02-02')  
    SELECT customer_id, visit_num from visits  
    WHERE page_view_dt='2016-02-02'  
    ORDER BY page_view_dt;
```

Each directory contains ORC files with two columns, customer_id and visit_num.

Accessing Partitioned Data from Vertica

Vertica recognizes and supports Hive-style partitions. You can read partition values and data using the HCatalog Connector or the COPY statement.

If you use the HCatalog Connector, you must create an HCatalog schema in Vertica that mirrors a schema in Hive:

```
=> CREATE EXTERNAL TABLE customer_visits (customer_id int, visit_num int,  
        page_view_dtm date)  
    AS COPY FROM 'hdfs://host:port/path/customer_visits/*/*' ORC  
    (hive_partition_cols='page_view_dtm');
```

The following statement reads all ORC files stored in all sub-directories including the partition values:

```
=> SELECT customer_id, visit_num, page_view FROM customer_visits;
```

When executing queries with predicates on partition columns, Vertica uses the subdirectory names to skip files that do not satisfy the predicate. This process is called *partition pruning*.

You can also define a separate external table for each subdirectory, as in the following example:

```
=> CREATE EXTERNAL TABLE customer_visits_20160201 (customer_id int,
        visit_num int, page_view_dtm date)
    AS COPY FROM
    'hdfs://host:port/path/customer_visits/page_view_dt=2016-02-01/*' ORC;
```

Example: A Partitioned, Sorted ORC Table

Suppose you have data stored in CSV files containing three columns: `customer_id`, `visit_num`, `page_view_dtm`:

```
1,123,2016-01-01
33,1,2016-02-01
2,57,2016-01-03
...
```

The goal is to create the following Hive table:

```
hive> CREATE TABLE customer_visits (
        customer_id bigint,
        visit_num int)
    PARTITIONED BY (page_view_dt date)
    STORED AS ORC;
```

To achieve this, perform the following steps:

1. Copy or move the CSV files to HDFS.
2. Define a textfile Hive table and copy the CSV files into it:

```
hive> CREATE TABLE visits (
        customer_id bigint,
        visit_num int,
        page_view_dt date)
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    STORED AS TEXTFILE;

hive> LOAD DATA INPATH path_to_csv_files INTO TABLE visits;
```

3. For each unique value in `page_view_dt`, insert the data into the target table while materializing `page_view_dt` as `page_view_dtm`:

```
hive> INSERT INTO TABLE customer_visits
      PARTITION (page_view_dt='2016-01-01')
      SELECT customer_id, visit_num FROM visits
      WHERE page_view_dt='2016-01-01'
      ORDER BY page_view_dt;
...
```

This operation inserts data from `visits.customer_id` into `customer_visits.customer_id`, and from `visits.visit_num` into `customer_visits.visit_num`. These two columns are stored in generated ORC files. Simultaneously, values from `visits.page_view_dt` are used to create partitions for the partition column `customer_visits.page_view_dt`, which is not stored in the ORC files.

Data Modification in Hive

Hive is well-suited for reading large amounts of write-once data. Its optimal usage is loading data in bulk into tables and never modifying the data. In particular, for data stored in the ORC and Parquet formats, this usage pattern produces large, globally (or nearly globally) sorted files.

Periodic addition of data to tables (known as “trickle load”) is likely to produce many small files. The disadvantage of this is that Vertica has to access many more files during query planning and execution. These extra access can result in longer query-processing time. The major performance degradation comes from the increase in the number of file seeks on HDFS.

Hive can also modify underlying ORC or Parquet files without user involvement. If enough records in a Hive table are modified or deleted, for example, Hive deletes existing files and replaces them with newly-created ones. Hive can also be configured to automatically merge many small files into a few larger files.

When new tables are created, or existing tables are modified in Hive, you must manually synchronize Vertica to keep it up to date. The following statement synchronizes the Vertica schema "hcat" after a change in Hive:

```
=> SELECT sync_with_hcatalog_schema('hcat_local', 'hcat');
```

Schema Evolution in Hive

Hive supports two kinds of schema evolution:

1. New columns can be added to existing tables in Hive. Vertica automatically handles this kind of schema evolution. The old records display NULLs for the newer columns.
2. The type of a column for a table can be modified in Hive. Vertica does not support this kind of schema evolution.

The following example demonstrates schema evolution through new columns. In this example, `hcat.parquet.txt` is a file with the following values:

```
-1|0.65|0.65|6|'b'
```

```
hive> create table hcat.parquet_tmp (a int, b float, c double, d int, e varchar(4))
      row format delimited fields terminated by '|' lines terminated by '\n';

hive> load data local inpath 'hcat.parquet.txt' overwrite into table
      hcat.parquet_tmp;

hive> create table hcat.parquet_evolve (a int) partitioned by (f int) stored as
      parquet;
hive> insert into table hcat.parquet_evolve partition (f=1) select a from
      hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (b float);
hive> insert into table hcat.parquet_evolve partition (f=2) select a, b from
      hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (c double);
hive> insert into table hcat.parquet_evolve partition (f=3) select a, b, c from
      hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (d int);
hive> insert into table hcat.parquet_evolve partition (f=4) select a, b, c, d from
      hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (e varchar(4));
hive> insert into table hcat.parquet_evolve partition (f=5) select a, b, c, d, e
      from hcat.parquet_tmp;
hive> insert into table hcat.parquet_evolve partition (f=6) select a, b, c, d, e
      from hcat.parquet_tmp;
```

```
=> SELECT * from hcat_local.parquet_evolve;
```

a	b	c	d	e	f
-1					1
-1	0.649999976158142				2
-1	0.649999976158142	0.65			3
-1	0.649999976158142	0.65	6		4
-1	0.649999976158142	0.65	6	b	5
-1	0.649999976158142	0.65	6	b	6

(6 rows)

Troubleshooting Reads from Native File Formats

You might encounter the following issues when reading ORC or Parquet files.

Reads from Parquet Files Report Unexpected Data-Type Mismatches

If a Parquet file contains a column of type `STRING` but the column in Vertica is of a different type, such as `INT`, you might see an unclear error message. In this case Vertica reports the column in the Parquet file as `BYTE_ARRAY`, as shown in the following example:

```
ERROR 0: Datatype mismatch: column 2 in the parquet_cpp source
[/tmp/nation.0.parquet] has type BYTE_ARRAY, expected int
```

This behavior is specific to Parquet files; with an ORC file the type is correctly reported as `STRING`. The problem occurs because Parquet does not natively support the `STRING` type and uses `BYTE_ARRAY` for strings instead. Because the Parquet file reports its type as `BYTE_ARRAY`, Vertica has no way to determine if the type is actually a `BYTE_ARRAY` or a `STRING`.

Time Zones in Timestamp Values Are Not Correct

Reading time stamps from an ORC or Parquet file in Vertica might result in different values, based on the local time zone. This issue occurs because the ORC and Parquet formats do not support the SQL `TIMESTAMP` data type. If you define the column in your table with the `TIMESTAMP` data type, Vertica interprets time stamps read from ORC or Parquet files as values in the local time zone. This same behavior occurs in Hive. When this situation occurs, Vertica produces a warning at query time, such as the following:

```
WARNING 0: SQL TIMESTAMPTZ is more appropriate for ORC TIMESTAMP
because values are stored in UTC
```

When creating the table in Vertica, you can avoid this issue by using the `TIMESTAMPTZ` data type instead of `TIMESTAMP`.

Some Date and Timestamp Values Are Wrong by Several Days

When Hive writes ORC or Parquet files, it converts dates before 1583 from the Gregorian calendar to the Julian calendar. Vertica does not perform this conversion. If your file contains dates before this time, values in Hive and the corresponding values in Vertica can differ by up to ten days. This difference applies to both DATE and TIMESTAMP values.

Error 7087: Wrong Number of Columns

When loading data, you might see an error stating that you have the wrong number of columns:

```
=> CREATE TABLE nation (nationkey bigint, name varchar(500),
                        regionkey bigint, comment varchar(500));
CREATE TABLE

=> COPY nation from :orc_dir ORC;
ERROR 7087: Attempt to load 4 columns from an orc source
[/tmp/orc_glob/test.orc] that has 9 columns
```

When you load data from Hadoop native file formats, your table must consume all of the data in the file, or this error results. To avoid this problem, add the missing columns to your table definition.

Error 7226: Cannot Find Partition Column

When querying data, you might see an error message stating that a partition column is missing:

```
ERROR 7226: Cannot find partition column [region] in parquet source
[/data/table_int/int_original/000000_0]
```

This error can occur if you partition your ORC or Parquet data (see [Using Partition Columns](#)). If you create an external table and then change the partition structure, for example by renaming a column, you must then re-create the external table. If you see this error, update your table to match the partitioning on disk.

Error 6766: Is a Directory

When querying data that is stored on the local disk, you might see an error message stating that an input file is a directory:

```
ERROR 6766: Error reading from orc parser input stream  
[/tmp/orc_glob/more_nations]: Is a directory
```

This error occurs if the glob in the table's COPY FROM clause matches an empty directory. This error occurs only for files in the Linux file system; empty directories in HDFS are ignored.

To correct the error, make the glob more specific. Instead of *, for example, use *.orc.

Exporting Data

You might want to export data from Vertica, either to share it with other Hadoop-based applications or to move lower-priority data from ROS to less-expensive storage. You can use the [EXPORT TO PARQUET](#) statement to export a table (or part of one) as Parquet data.

You can export data to HDFS or to the local file system. You can export ROS data or data that is readable through external tables. After exporting ROS data, you can drop affected ROS partitions to reclaim storage space. If you need to access the data in Vertica again, you can create external tables from the exported data.

To export data, use EXPORT TO PARQUET in combination with a SELECT statement, as in the following example:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_data')
  AS SELECT * FROM public.sales;
  Rows Exported
  -----
                14336
(1 row)
```

The directory argument specifies where to write the files and is required. The directory must not already exist.

You can use EXPORT TO PARQUET to write queries across multiple tables in Vertica and export the results. With this approach you can take advantage of powerful, fast query execution in Vertica while making the results available to other Hadoop clients:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_by_region')
  AS SELECT sale.price, sale.date, store.region
  FROM public.sales sale
  JOIN public.vendor store ON sale.distribID = store.ID;
  Rows Exported
  -----
                23301
(1 row)
```

EXPORT TO PARQUET takes optional parameters to specify compression format and row-group size (in MB), as in the following example:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_data',
                    compression = 'uncompressed', rowGroupSize = '32')
  AS SELECT * FROM public.sales;
  Rows Exported
  -----
                14336
(1 row)
```

The default compression type is Snappy.

The row-group size affects memory consumption during export. An export thread consumes at least 64MB of RAM if the value is 64. The default value of 64 is a compromise between writing larger row groups and allowing enough free memory for other Vertica operations. If you perform exports when the database is not otherwise under heavy load, you can improve read performance later by increasing row-group size on export.

When exporting, you can use the optional `OVER` clause to specify how to partition and/or sort data. Partitioning reduces the sizes of the output data files and can improve performance when Vertica queries external tables containing this data. (See [Using Partition Columns](#).) If you do not specify how to partition the data, Vertica optimizes the export for maximum parallelism.

To specify partition columns, use `PARTITION BY` in the `OVER` clause as in the following example:

```
=> EXPORT TO PARQUET(directory = 'hdfs:///data/export')
  OVER(PARTITION BY date) AS SELECT date, price FROM public.sales;
Rows Exported
-----
          28337
(1 row)
```

You can sort values within each partition for a further performance improvement. Sort table columns based on the likelihood of their occurrence in query predicates; columns that most frequently occur in comparison or range predicates should be sorted first. You can sort values within each partition using `ORDER BY` in the `OVER` clause:

```
=> EXPORT TO PARQUET(directory = 'hdfs:///data/export')
  OVER(PARTITION BY date ORDER BY price) AS SELECT date, price FROM public.sales;
Rows Exported
-----
          28337
(1 row)
```

You can use `ORDER BY` even without partitioning. Storing data in sorted order can improve data access and predicate evaluation performance.

Targets in the `OVER` clause must be column references; they cannot be expressions. For more information about `OVER`, see [SQL Analytics](#).

If you are exporting data to a local file system, you might want to force a single node to write all of the files. To do so, use an empty `OVER` clause.

You cannot export columns with the `TIME`, `TIMEZ`, and `INTERVAL` data types. If your table includes columns of these types, exclude them by explicitly selecting the columns you can export:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_data')
  AS SELECT date, transactionID, price FROM public.sales;
Rows Exported
-----
          14336
```

(1 row)

You can only perform one export per output directory. If you perform more than one concurrent export to the same directory, only one will succeed.

Using the HCatalog Connector

The Vertica HCatalog Connector lets you access data stored in Apache's Hive data warehouse software the same way you access it within a native Vertica table.

If your files are in the Optimized Columnar Row (ORC) or Parquet format and do not use complex types, the HCatalog Connector creates an external table and uses the ORC or Parquet reader instead of using the Java SerDe. See [Reading Hadoop Columnar File Formats](#) for more information about these readers.

The HCatalog Connector performs predicate pushdown to improve query performance. Instead of reading all data across the network to evaluate a query, the HCatalog Connector moves the evaluation of predicates closer to the data. Predicate pushdown applies to Hive partition pruning, ORC stripe pruning, and Parquet row-group pruning. The HCatalog Connector supports predicate pushdown for the following predicates: $>$, $>=$, $=$, $<>$, $<=$, $<$.

Hive, HCatalog, and HiveServer2 Overview

There are several Hadoop components that you need to understand to use the HCatalog connector:

- Apache's Hive lets you query data stored in a Hadoop Distributed File System (HDFS) the same way you query data stored in a relational database. Behind the scenes, Hive uses a set of serializer and deserializer (SerDe) classes to extract data from files stored in HDFS and break it into columns and rows. Each SerDe handles data files in a specific format. For example, one SerDe extracts data from comma-separated data files while another interprets data stored in JSON format.
- Apache HCatalog is a component of the Hadoop ecosystem that makes Hive's metadata available to other Hadoop components (such as Pig).
- HiveServer2 makes HCatalog and Hive data available via JDBC. Through it, a client can make requests to retrieve data stored in Hive, as well as information about the Hive schema.

The Vertica HCatalog Connector lets you transparently access data that is available through HiveServer2. You use the connector to define a schema in Vertica that corresponds to a Hive database or schema. When you query data within this schema, the HCatalog Connector transparently extracts and formats the data from Hadoop into tabular data.

Note: You can use the WebHCat service instead of HiveServer2, but performance is usually better with HiveServer2. To use WebHCat, set the `HCatalogConnectorUseHiveServer2` configuration parameter to 0. See [Hadoop Parameters](#).

HCatalog Connection Features

The HCatalog Connector lets you query data stored in Hive using the Vertica native SQL syntax. Some of its main features are:

- The HCatalog Connector always reflects the current state of data stored in Hive.
- The HCatalog Connector uses the parallel nature of both Vertica and Hadoop to process Hive data. The result is that querying data through the HCatalog Connector is often faster than querying the data directly through Hive.
- Because Vertica performs the extraction and parsing of data, the HCatalog Connector does not significantly increase the load on your Hadoop cluster.
- The data you query through the HCatalog Connector can be used as if it were native Vertica data. For example, you can execute a query that joins data from a table in an HCatalog schema with a native table.

HCatalog Connector Considerations

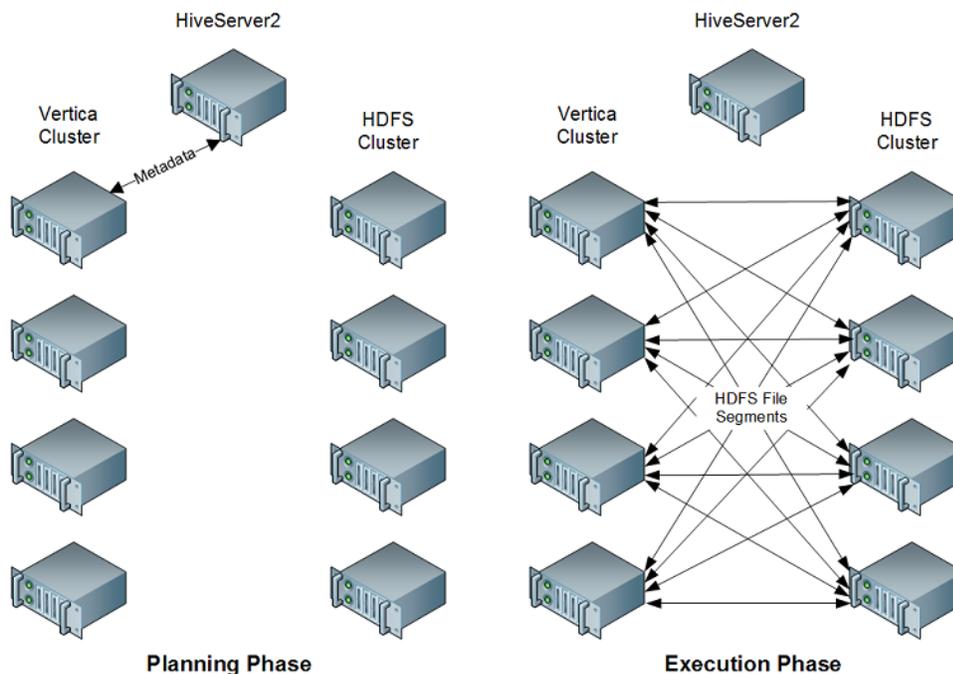
There are a few things to keep in mind when using the HCatalog Connector:

- Hive's data is stored in flat files in a distributed filesystem, requiring it to be read and deserialized each time it is queried. This deserialization causes Hive performance to be much slower than that of Vertica. The HCatalog Connector has to perform the same process as Hive to read the data. Therefore, querying data stored in Hive using the HCatalog Connector is much slower than querying a native Vertica table. If you need to perform extensive analysis on data stored in Hive, you should consider loading it into Vertica. Vertica optimization often makes querying data through the HCatalog Connector faster than directly querying it through Hive.
- Hive supports complex data types such as lists, maps, and structs that Vertica does not support. Columns containing these data types are converted to a JSON representation of the data type and stored as a VARCHAR. See [Data Type Conversions from Hive to Vertica](#).

Note: The HCatalog Connector is read-only. It cannot insert data into Hive.

How the HCatalog Connector Works

When planning a query that accesses data from a Hive table, the Vertica HCatalog Connector on the initiator node contacts HiveServer2 (or WebHCat) in your Hadoop cluster to determine if the table exists. If it does, the connector retrieves the table's metadata from the metastore database so the query planning can continue. When the query executes, all nodes in the Vertica cluster directly retrieve the data necessary for completing the query from HDFS. They then use the Hive SerDe classes to extract the data so the query can execute. When accessing data in ORC or Parquet format, the HCatalog Connector uses the readers for these formats instead of the Hive SerDe classes.



This approach takes advantage of the parallel nature of both Vertica and Hadoop. In addition, by performing the retrieval and extraction of data directly, the HCatalog Connector reduces the impact of the query on the Hadoop cluster.

HCatalog Connector Requirements

Before you can use the HCatalog Connector, both your Vertica and Hadoop installations must meet the following requirements.

Vertica Requirements

All of the nodes in your cluster must have a Java Virtual Machine (JVM) installed. You must use the same Java version that the Hadoop cluster uses. See [Installing the Java Runtime on Your Vertica Cluster](#).

You must also add certain libraries distributed with Hadoop and Hive to your Vertica installation directory. See [Configuring Vertica for HCatalog](#).

Hadoop Requirements

Your Hadoop cluster must meet several requirements to operate correctly with the Vertica Connector for HCatalog:

- It must have Hive, HiveServer2, and HCatalog installed and running. See Apache's [HCatalog](#) page for more information.
- The HiveServer2 server and all of the HDFS nodes that store HCatalog data must be directly accessible from all of the hosts in your Vertica database. Verify that any firewall separating the Hadoop cluster and the Vertica cluster will pass HiveServer2, metastore database, and HDFS traffic.
- The data that you want to query must be in an internal or external Hive table.
- When using the ORC and Parquet readers through the HCatalog Connector, partitioned data must be located in the default file location. If you have moved partitions, set the Vertica configuration parameters `HCatalogConnectorUseORCReader` and `HCatalogConnectorUseParquetReader` to 0 to use the Java UDX for these formats. Querying ORC and Parquet data through the Java UDX is slower than using the built-in readers.
- If a table you want to query uses a non-standard SerDe, you must install the SerDe's classes on your Vertica cluster before you can query the data. See [Using Nonstandard SerDes](#).

Installing the Java Runtime on Your Vertica Cluster

The HCatalog Connector requires a 64-bit Java Virtual Machine (JVM). The JVM must support Java 6 or later, and must be the same version as the one installed on your Hadoop nodes.

Note: If your Vertica cluster is configured to execute User Defined Extensions (UDxs) written in Java, it already has a correctly-configured JVM installed. See [Developing User-Defined Extensions \(UDxs\)](#) in Extending Vertica for more information.

Installing Java on your Vertica cluster is a two-step process:

1. Install a Java runtime on all of the hosts in your cluster.
2. Set the `JavaBinaryForUDx` configuration parameter to tell Vertica the location of the Java executable.

Installing a Java Runtime

For Java-based features, Vertica requires a 64-bit Java 6 (Java version 1.6) or later Java runtime. Vertica supports runtimes from either Oracle or [OpenJDK](#). You can choose to install either the Java Runtime Environment (JRE) or Java Development Kit (JDK), since the JDK also includes the JRE.

Many Linux distributions include a package for the OpenJDK runtime. See your Linux distribution's documentation for information about installing and configuring OpenJDK.

To install the Oracle Java runtime, see the [Java Standard Edition \(SE\) Download Page](#). You usually run the installation package as root in order to install it. See the download page for instructions.

Once you have installed a JVM on each host, ensure that the `java` command is in the search path and calls the correct JVM by running the command:

```
$ java -version
```

This command should print something similar to:

```
java version "1.8.0_102"  
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
```

Note: Any previously installed Java VM on your hosts may interfere with a newly installed Java runtime. See your Linux distribution's documentation for instructions on configuring which JVM is the default. Unless absolutely required, you should uninstall any incompatible version of Java before installing the Java 6 or Java 7 runtime.

Setting the JavaBinaryForUDx Configuration Parameter

The JavaBinaryForUDx configuration parameter tells Vertica where to look for the JRE to execute Java UDxs. After you have installed the JRE on all of the nodes in your cluster, set this parameter to the absolute path of the Java executable. You can use the symbolic link that some Java installers create (for example `/usr/bin/java`). If the Java executable is in your shell search path, you can get the path of the Java executable by running the following command from the Linux command line shell:

```
$ which java
/usr/bin/java
```

If the `java` command is not in the shell search path, use the path to the Java executable in the directory where you installed the JRE. Suppose you installed the JRE in `/usr/java/default` (which is where the installation package supplied by Oracle installs the Java 1.6 JRE). In this case the Java executable is `/usr/java/default/bin/java`.

You set the configuration parameter by executing the following statement as a database superuser:

```
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
```

See [ALTER DATABASE](#) for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the [CONFIGURATION_PARAMETERS](#) system table:

```
=> \x
Expanded display is on.
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'JavaBinaryForUDx';
-[ RECORD 1 ]-----+-----
node_name          | ALL
parameter_name     | JavaBinaryForUDx
current_value      | /usr/bin/java
default_value      |
change_under_support_guidance | f
change_requires_restart | f
description        | Path to the java binary for executing UDx written in Java
```

Once you have set the configuration parameter, Vertica can find the Java executable on each node in your cluster.

Note: Since the location of the Java executable is set by a single configuration parameter for the entire cluster, you must ensure that the Java executable is installed in the same path on all of the hosts in the cluster.

Configuring Vertica for HCatalog

Before you can use the HCatalog Connector, you must add certain Hadoop and Hive libraries to your Vertica installation. You must also copy the Hadoop configuration files that specify various connection properties. Vertica uses the values in those configuration files to make its own connections to Hadoop.

You need only make these changes on one node in your cluster. After you do this you can install the HCatalog connector.

Copy Hadoop Libraries and Configuration Files

Vertica provides a tool, `hcatUtil`, to collect the required files from Hadoop. This tool copies selected libraries and XML configuration files from your Hadoop cluster to your Vertica cluster. This tool might also need access to additional libraries:

- If you plan to use Hive to query files that use Snappy compression, you need access to the Snappy native libraries, `libhadoop*.so` and `libsnapappy*.so`.
- If you plan to use Hive to query files that use LZO compression, you need access to the `hadoop-lzo-*.jar` and `libgplcompression.so*` libraries. In `core-site.xml` you must also edit the `io.compression.codecs` property to include `com.hadoop.compression.lzo.LzopCodec`.
- If you plan to use a JSON SerDe with a Hive table, you need access to its library. This is the same library that you used to configure Hive; for example:

```
hive> add jar /home/release/json-serde-1.3-jar-with-dependencies.jar;

hive> create external table nationjson (id int,name string,rank int,text string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION '/user/release/vt/nationjson';
```

- If you are using any other libraries that are not standard across all supported Hadoop versions, you need access to those libraries.

If any of these cases applies to you, do one of the following:

- Include the path(s) in the path you specify as the value of `--hcatLibPath`, or
- Copy the file(s) to a directory already on that path.

If Vertica is not co-located on a Hadoop node, you should do the following:

1. Copy `/opt/vertica/packages/hcat/tools/hcatUtil` to a Hadoop node and run it there, specifying a temporary output directory. Your Hadoop, HIVE, and HCatalog lib paths might be different. In newer versions of Hadoop the HCatalog directory is usually a subdirectory under the HIVE directory, and Cloudera creates a new directory for each revision of the configuration files. Use the values from your environment in the following command:

```
hcatUtil --copyJars
--hadoopHiveHome="$HADOOP_HOME/lib;$HIVE_HOME/lib;/hcatalog/dist/share"
--hadoopHiveConfPath="$HADOOP_CONF_DIR;$HIVE_CONF_DIR;$WEBHCAT_CONF_DIR"
--hcatLibPath="/tmp/hadoop-files"
```

2. Verify that all necessary files were copied:

```
hcatUtil --verifyJars --hcatLibPath=/tmp/hadoop-files
```

3. Copy that output directory (`/tmp/hadoop-files`, in this example) to `/opt/vertica/packages/hcat/lib` on the Vertica node you will connect to when installing the HCatalog connector. If you are updating a Vertica cluster to use a new Hadoop cluster (or a new version of Hadoop), first remove all JAR files in `/opt/vertica/packages/hcat/lib` except `vertica-hcatalogudl.jar`.

4. Verify that all necessary files were copied:

```
hcatUtil --verifyJars --hcatLibPath=/opt/vertica/packages/hcat
```

If Vertica is co-located on some or all Hadoop nodes, you can do this in one step on a shared node. Your Hadoop, HIVE, and HCatalog lib paths might be different; use the values from your environment in the following command:

```
hcatUtil --copyJars
--hadoopHiveHome="$HADOOP_HOME/lib;$HIVE_HOME/lib;/hcatalog/dist/share"
--hadoopHiveConfPath="$HADOOP_CONF_DIR;$HIVE_CONF_DIR;$WEBHCAT_CONF_DIR"
--hcatLibPath="/opt/vertica/packages/hcat/lib"
```

The `hcatUtil` script has the following arguments:

<p><code>-c, --copyJars</code></p>	<p>Copy the required JAR files from <code>hadoopHiveHome</code> and configuration files from <code>hadoopHiveConfPath</code>.</p>
<p><code>-v, --verifyJars</code></p>	<p>Verify that the required files are present in <code>hcatLibPath</code>. Check the output of <code>hcatUtil</code> for error and warning messages.</p>
<p><code>--hadoopHiveHome="value1;value2;..."</code></p>	<p>Paths to the Hadoop, Hive, and HCatalog home directories. Separate directories by semicolons (;). Enclose paths in double quotes.</p> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <p>Note: Always place <code>\$HADOOP_HOME</code> on the path before <code>\$HIVE_HOME</code>. In some Hadoop distributions, these two directories contain different versions of the same library.</p> </div>
<p><code>--hadoopHiveConfPath="value1;value2;..."</code></p>	<p>Paths of the following configuration files:</p> <ul style="list-style-type: none"> • <code>hive-site.xml</code> • <code>core-site.xml</code> • <code>yarn-site.xml</code> • <code>webhcat-site.xml</code> (optional with the default configuration; required if you use WebHCat instead of HiveServer2) • <code>hdfs-site.xml</code> <p>Separate directories by semicolons (;). Enclose paths in double quotes.</p> <p>In previous releases of Vertica this parameter was optional under some conditions. It is now required.</p>
<p><code>--hcatLibPath="value"</code></p>	<p>Output path for the libraries and configuration files. On a Vertica node, use <code>/opt/vertica/packages/hcat/lib</code>. If you have previously run <code>hcatUtil</code> with a different version of Hadoop, first remove the old JAR files from the output directory (all except <code>vertica-hcatalogudl.jar</code>).</p>

After you have copied the files and verified them, install the HCatalog connector.

Install the HCatalog Connector

On the same node where you copied the files from `hcatUtil`, install the HCatalog connector by running the `install.sql` script. This script resides in the `ddl/` folder under your HCatalog connector installation path. This script creates the library and `VHCatSource` and `VHCatParser`.

Note: The data that was copied using `hcatUtil` is now stored in the database. If you change any of those values in Hadoop, you need to rerun `hcatUtil` and `install.sql`. The following statement returns the names of the libraries and configuration files currently being used:

```
=> SELECT dependencies FROM user_libraries WHERE lib_name='VHCatalogLib';
```

Now you can create HCatalog schema parameters, which point to your existing Hadoop services, as described in [Defining a Schema Using the HCatalog Connector](#).

Upgrading to a New Version of Vertica

After upgrading to a new version of Vertica, perform the following steps:

1. Uninstall the HCatalog Connector using the `uninstall.sql` script. This script resides in the `ddl/` folder under your HCatalog connector installation path.
2. Delete the contents of the `hcatLibPath` directory except for `vertica-hcatalogudl.jar`.
3. Rerun `hcatUtil`.
4. Reinstall the HCatalog Connector using the `install.sql` script.

For more information about upgrading Vertica, see [Upgrade Vertica](#).

Additional Options for Hadoop Columnar File Formats

When reading Hadoop columnar file formats (ORC or Parquet), the HCatalog Connector attempts to use the built-in readers. When doing so, it uses the `hdfs` scheme by default. In order to use the `hdfs` scheme, you must perform the configuration described in [Configuring the hdfs Scheme](#).

To have the HCatalog Connector use the `webhdfs` scheme instead, use `ALTER DATABASE` to set `HCatalogConnectorUseLibHDFSPP` to 0.

Using the HCatalog Connector with HA NameNode

Newer distributions of Hadoop support the High Availability NameNode (HA NN) for HDFS access. Some additional configuration is required to use this feature with the HCatalog Connector. If you do not perform this configuration, attempts to retrieve data through the connector will produce an error.

To use HA NN with Vertica, first copy `/etc/hadoop/conf` from the HDFS cluster to every node in your Vertica cluster. You can put this directory anywhere, but it must be in the same location on every node. (In the example below it is in `/opt/hcat/hadoop_conf`.)

Then uninstall the HCat library, configure the UDX to use that configuration directory, and reinstall the library:

```
=> \i /opt/vertica/packages/hcat/ddl/uninstall.sql
      DROP LIBRARY

=> ALTER DATABASE mydb SET JavaClassPathSuffixForUDx = '/opt/hcat/hadoop_conf';
      WARNING 2693: Configuration parameter JavaClassPathSuffixForUDx has been deprecated;
                  setting it has no effect

=> \i /opt/vertica/packages/hcat/ddl/install.sql
      CREATE LIBRARY
      CREATE SOURCE FUNCTION
      GRANT PRIVILEGE
      CREATE PARSER FUNCTION
      GRANT PRIVILEGE
```

Despite the warning message, this step is necessary.

After taking these steps, HCatalog queries will now work.

Defining a Schema Using the HCatalog Connector

After you set up the HCatalog Connector, you can use it to define a schema in your Vertica database to access the tables in a Hive database. You define the schema using the [CREATE HCATALOG SCHEMA](#) statement.

When creating the schema, you must supply the name of the schema to define in Vertica. Other parameters are optional. If you do not supply a value, Vertica uses default values. Vertica reads some default values from the HDFS configuration files.

After you define the schema, you can query the data in the Hive data warehouse in the same way you query a native Vertica table. The following example demonstrates creating an HCatalog schema and then querying several system tables to examine the contents of the new schema. See [Viewing Hive Schema and Table Metadata](#) for more information about these tables.

```
=> CREATE HCATALOG SCHEMA hcat WITH HOSTNAME='hcatost'
      HCATALOG_SCHEMA='default' HIVESERVER2_HOSTNAME='hs.example.com'
      HCATALOG_USER='admin';
WARNING 0:  HOSTNAME will be ignored. hive-site.xml must be contain property [hive.metastore.uris],
or both HOSTNAME and PORT must be specified
CREATE SCHEMA
=> \x
Expanded display is on.

=> SELECT * FROM v_catalog.hcatalog_schemata;
-[ RECORD 1 ]-----+-----
schema_id          | 45035996273748224
schema_name        | hcat
schema_owner_id    | 45035996273704962
schema_owner       | admin
create_time        | 2017-05-22 12:04:59.692155-04
hostname           | hcatost
port               | -1
hiveserver2_hostname | hs.example.com
webservice_hostname |
webservice_port    | 50111
webhdfs_address    | hs.example.com:50070
hcatalog_schema_name | default
hcatalog_user_name  | admin
hcatalog_connection_timeout | -1
hcatalog_slow_transfer_limit | -1
hcatalog_slow_transfer_time | -1

=> SELECT * FROM v_catalog.hcatalog_table_list;
-[ RECORD 1 ]-----+-----
table_schema_id    | 45035996273748224
table_schema       | hcat
hcatalog_schema    | default
table_name         | nation
hcatalog_user_name | admin
-[ RECORD 2 ]-----+-----
table_schema_id    | 45035996273748224
table_schema       | hcat
hcatalog_schema    | default
table_name         | raw
hcatalog_user_name | admin
-[ RECORD 3 ]-----+-----
table_schema_id    | 45035996273748224
table_schema       | hcat
hcatalog_schema    | default
table_name         | raw_rcfile
hcatalog_user_name | admin
```

```
-[ RECORD 4 ]-----+-----  
table_schema_id | 45035996273748224  
table_schema    | hcat  
hcatalog_schema | default  
table_name      | raw_sequence  
hcatalog_user_name | admin
```

Querying Hive Tables Using HCatalog Connector

Once you have defined the HCatalog schema, you can query data from the Hive database by using the schema name in your query.

```
=> SELECT * from hcat.messages limit 10;  
messageid | userid | time | message  
-----+-----+-----+-----  
1 | nPfQ1ayhi | 2013-10-29 00:10:43 | hymenaeos cursus lorem Suspendis  
2 | N7svORIoZ | 2013-10-29 00:21:27 | Fusce ad sem vehicula morbi  
3 | 4VvzN3d | 2013-10-29 00:32:11 | porta Vivamus condimentum  
4 | heojkmTmc | 2013-10-29 00:42:55 | lectus quis imperdiet  
5 | coROws30F | 2013-10-29 00:53:39 | sit eleifend tempus a aliquam mauri  
6 | oDRP1i | 2013-10-29 01:04:23 | risus facilisis sollicitudin sceler  
7 | AU7a9Kp | 2013-10-29 01:15:07 | turpis vehicula tortor  
8 | ZJWg185DkZ | 2013-10-29 01:25:51 | sapien adipiscing eget Aliquam tor  
9 | E7ipAsYc3 | 2013-10-29 01:36:35 | varius Cum iaculis metus  
10 | kStCv | 2013-10-29 01:47:19 | aliquam libero nascetur Cum mal  
(10 rows)
```

Since the tables you access through the HCatalog Connector act like Vertica tables, you can perform operations that use both Hive data and native Vertica data, such as a join:

```
=> SELECT u.FirstName, u.LastName, d.time, d.Message from UserData u  
-> JOIN hcat.messages d ON u.UserID = d.UserID LIMIT 10;  
FirstName | LastName | time | Message  
-----+-----+-----+-----  
Whitney | Kerr | 2013-10-29 00:10:43 | hymenaeos cursus lorem Suspendis  
Troy | Oneal | 2013-10-29 00:32:11 | porta Vivamus condimentum  
Renee | Coleman | 2013-10-29 00:42:55 | lectus quis imperdiet  
Fay | Moss | 2013-10-29 00:53:39 | sit eleifend tempus a aliquam mauri  
Dominique | Cabrera | 2013-10-29 01:15:07 | turpis vehicula tortor  
Mohammad | Eaton | 2013-10-29 00:21:27 | Fusce ad sem vehicula morbi  
Cade | Barr | 2013-10-29 01:25:51 | sapien adipiscing eget Aliquam tor  
Oprah | Mcmillan | 2013-10-29 01:36:35 | varius Cum iaculis metus  
Astra | Sherman | 2013-10-29 01:58:03 | dignissim odio Pellentesque primis  
Chelsea | Malone | 2013-10-29 02:08:47 | pede tempor dignissim Sed luctus  
(10 rows)
```

Viewing Hive Schema and Table Metadata

When using Hive, you access metadata about schemas and tables by executing statements written in HiveQL (Hive's version of SQL) such as `SHOW TABLES`. When using the HCatalog Connector, you can get metadata about the tables in the Hive database through several Vertica system tables.

There are four system tables that contain metadata about the tables accessible through the HCatalog Connector:

- [HCATALOG_SCHEMATA](#) lists all of the schemas that have been defined using the HCatalog Connector.
- [HCATALOG_TABLE_LIST](#) contains an overview of all of the tables available from all schemas defined using the HCatalog Connector. This table only shows the tables that the user querying the table can access. The information in this table is retrieved using a single call to HiveServer2 for each schema defined using the HCatalog Connector, which means there is a little overhead when querying this table.
- [HCATALOG_TABLES](#) contains more in-depth information than [HCATALOG_TABLE_LIST](#).
- [HCATALOG_COLUMNS](#) lists metadata about all of the columns in all of the tables available through the HCatalog Connector. As for [HCATALOG_TABLES](#), querying this table results in one call to HiveServer2 per table, and therefore can take a while to complete.

The following example demonstrates querying the system tables containing metadata for the tables available through the HCatalog Connector.

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcat'
-> HCATALOG_SCHEMA='default' HCATALOG_DB='default' HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT * FROM HCATALOG_SCHEMATA;
-[ RECORD 1 ]-----+-----
schema_id          | 45035996273864536
schema_name        | hcat
schema_owner_id   | 45035996273704962
schema_owner       | dbadmin
create_time        | 2013-11-05 10:19:54.70965-05
hostname           | hcat
port               | 9083
webservice_hostname | hcat
webservice_port    | 50111
hcatalog_schema_name | default
hcatalog_user_name  | hcatuser
metastore_db_name  | hivesmetastoredb
```

```
=> SELECT * FROM HCATALOG_TABLE_LIST;
-[ RECORD 1 ]-----+-----
table_schema_id | 45035996273864536
table_schema    | hcat
hcatalog_schema | default
table_name      | hcatalogtypes
hcatalog_user_name | hcatuser
-[ RECORD 2 ]-----+-----
table_schema_id | 45035996273864536
table_schema    | hcat
hcatalog_schema | default
table_name      | tweets
hcatalog_user_name | hcatuser
-[ RECORD 3 ]-----+-----
table_schema_id | 45035996273864536
table_schema    | hcat
hcatalog_schema | default
table_name      | messages
hcatalog_user_name | hcatuser
-[ RECORD 4 ]-----+-----
table_schema_id | 45035996273864536
table_schema    | hcat
hcatalog_schema | default
table_name      | msgjson
hcatalog_user_name | hcatuser

=> -- Get detailed description of a specific table
=> SELECT * FROM HCATALOG_TABLES WHERE table_name = 'msgjson';
-[ RECORD 1 ]-----+-----
table_schema_id | 45035996273864536
table_schema    | hcat
hcatalog_schema | default
table_name      | msgjson
hcatalog_user_name | hcatuser
min_file_size_bytes |
total_number_files | 10
location         | hdfs://hive.example.com:8020/user/exampleuser/msgjson
last_update_time |
output_format    | org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat
last_access_time |
max_file_size_bytes |
is_partitioned   | f
partition_expression |
table_owner      |
input_format     | org.apache.hadoop.mapred.TextInputFormat
total_file_size_bytes | 453534
hcatalog_group   |
permission       |

=> -- Get list of columns in a specific table
=> SELECT * FROM HCATALOG_COLUMNS WHERE table_name = 'hcatalogtypes'
-> ORDER BY ordinal_position;
-[ RECORD 1 ]-----+-----
table_schema    | hcat
hcatalog_schema | default
table_name      | hcatalogtypes
is_partition_column | f
column_name     | intcol
hcatalog_data_type | int
data_type       | int
```

```

data_type_id          | 6
data_type_length      | 8
character_maximum_length |
numeric_precision     |
numeric_scale         |
datetime_precision   |
interval_precision   |
ordinal_position      | 1
-[ RECORD 2 ]-----+-----
table_schema          | hcat
hcatalog_schema       | default
table_name            | hcatalogtypes
is_partition_column   | f
column_name           | floatcol
hcatalog_data_type    | float
data_type             | float
data_type_id          | 7
data_type_length      | 8
character_maximum_length |
numeric_precision     |
numeric_scale         |
datetime_precision   |
interval_precision   |
ordinal_position      | 2
-[ RECORD 3 ]-----+-----
table_schema          | hcat
hcatalog_schema       | default
table_name            | hcatalogtypes
is_partition_column   | f
column_name           | doublecol
hcatalog_data_type    | double
data_type             | float
data_type_id          | 7
data_type_length      | 8
character_maximum_length |
numeric_precision     |
numeric_scale         |
datetime_precision   |
interval_precision   |
ordinal_position      | 3
-[ RECORD 4 ]-----+-----
table_schema          | hcat
hcatalog_schema       | default
table_name            | hcatalogtypes
is_partition_column   | f
column_name           | charcol
hcatalog_data_type    | string
data_type             | varchar(65000)
data_type_id          | 9
data_type_length      | 65000
character_maximum_length | 65000
numeric_precision     |
numeric_scale         |
datetime_precision   |
interval_precision   |
ordinal_position      | 4
-[ RECORD 5 ]-----+-----
table_schema          | hcat
hcatalog_schema       | default
table_name            | hcatalogtypes

```

```

is_partition_column | f
column_name         | varcharcol
hcatalog_data_type  | string
data_type           | varchar(65000)
data_type_id        | 9
data_type_length    | 65000
character_maximum_length | 65000
numeric_precision   |
numeric_scale       |
datetime_precision |
interval_precision |
ordinal_position    | 5
-[ RECORD 6 ]-----+-----
table_schema        | hcat
hcatalog_schema     | default
table_name          | hcatalogtypes
is_partition_column | f
column_name         | boolcol
hcatalog_data_type  | boolean
data_type           | boolean
data_type_id        | 5
data_type_length    | 1
character_maximum_length |
numeric_precision   |
numeric_scale       |
datetime_precision |
interval_precision |
ordinal_position    | 6
-[ RECORD 7 ]-----+-----
table_schema        | hcat
hcatalog_schema     | default
table_name          | hcatalogtypes
is_partition_column | f
column_name         | timestampcol
hcatalog_data_type  | string
data_type           | varchar(65000)
data_type_id        | 9
data_type_length    | 65000
character_maximum_length | 65000
numeric_precision   |
numeric_scale       |
datetime_precision |
interval_precision |
ordinal_position    | 7
-[ RECORD 8 ]-----+-----
table_schema        | hcat
hcatalog_schema     | default
table_name          | hcatalogtypes
is_partition_column | f
column_name         | varbincol
hcatalog_data_type  | binary
data_type           | varbinary(65000)
data_type_id        | 17
data_type_length    | 65000
character_maximum_length | 65000
numeric_precision   |
numeric_scale       |
datetime_precision |
interval_precision |
ordinal_position    | 8

```

```
-[ RECORD 9 ]-----+-----  
table_schema          | hcat  
hcatalog_schema      | default  
table_name           | hcatalogtypes  
is_partition_column  | f  
column_name          | bincol  
hcatalog_data_type   | binary  
data_type            | varbinary(65000)  
data_type_id         | 17  
data_type_length     | 65000  
character_maximum_length | 65000  
numeric_precision   |  
numeric_scale        |  
datetime_precision  |  
interval_precision  |  
ordinal_position     | 9
```

Synchronizing an HCatalog Schema or Table With a Local Schema or Table

Querying data from an HCatalog schema can be slow due to Hive performance issues. This slow performance can be especially annoying when you want to examine the structure of the tables in the Hive database. Getting this information from Hive requires you to query the HCatalog schema's metadata using the HCatalog Connector.

To avoid this performance problem you can use the [SYNC_WITH_HCATALOG_SCHEMA](#) function to create a snapshot of the HCatalog schema's metadata within a Vertica schema. You supply this function with the name of a pre-existing Vertica schema, typically the one created through `CREATE HCATALOG SCHEMA`, and a Hive schema available through the HCatalog Connector. The function creates a set of external tables within the Vertica schema that you can then use to examine the structure of the tables in the Hive database. Because the metadata in the Vertica schema is local, query planning is much faster. You can also use standard Vertica statements and system-table queries to examine the structure of Hive tables in the HCatalog schema.

Caution: The `SYNC_WITH_HCATALOG_SCHEMA` function overwrites tables in the Vertica schema whose names match a table in the HCatalog schema. Do not use the Vertica schema to store other data.

When `SYNC_WITH_HCATALOG_SCHEMA` creates tables in Vertica, it matches Hive's `STRING` and `BINARY` types to Vertica's `VARCHAR(65000)` and `VARBINARY(65000)` types. You might want to change these lengths, using [ALTER TABLE SET DATA TYPE](#), in two cases:

- If the value in Hive is larger than 65000 bytes, increase the size and use LONG VARCHAR or LONG VARBINARY to avoid data truncation. If a Hive string uses multi-byte encodings, you must increase the size in Vertica to avoid data truncation. This step is needed because Hive counts string length in characters while Vertica counts it in bytes.
- If the value in Hive is much smaller than 65000 bytes, reduce the size to conserve memory in Vertica.

The Vertica schema is just a snapshot of the HCatalog schema's metadata. Vertica does not synchronize later changes to the HCatalog schema with the local schema after you call `SYNC_WITH_HCATALOG_SCHEMA`. You can call the function again to re-synchronize the local schema to the HCatalog schema. If you altered column data types, you will need to repeat those changes because the function creates new external tables.

By default, `SYNC_WITH_HCATALOG_SCHEMA` does not drop tables that appear in the local schema that do not appear in the HCatalog schema. Thus, after the function call the local schema does not reflect tables that have been dropped in the Hive database since the previous call. You can change this behavior by supplying the optional third Boolean argument that tells the function to drop any table in the local schema that does not correspond to a table in the HCatalog schema.

Instead of synchronizing the entire schema, you can synchronize individual tables by using [SYNC_WITH_HCATALOG_SCHEMA_TABLE](#). If the table already exists in Vertica the function overwrites it. If the table is not found in the HCatalog schema, this function returns an error. In all other respects this function behaves in the same way as `SYNC_WITH_HCATALOG_SCHEMA`.

If you change the settings of any HCatalog Connector configuration parameters ([Hadoop Parameters](#)), you must call this function again.

Examples

The following example demonstrates calling `SYNC_WITH_HCATALOG_SCHEMA` to synchronize the HCatalog schema in Vertica with the metadata in Hive. Because it synchronizes the HCatalog schema directly, instead of synchronizing another schema with the HCatalog schema, both arguments are the same.

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcatost' HCATALOG_SCHEMA='default'  
-> HCATALOG_USER='hcatuser';  
CREATE SCHEMA  
=> SELECT sync_with_hcatalog_schema('hcat', 'hcat');  
sync_with_hcatalog_schema  
-----  
Schema hcat synchronized with hcat  
tables in hcat = 56
```

```

tables altered in hcat = 0
tables created in hcat = 56
stale tables in hcat = 0
table changes erred in hcat = 0
(1 row)

=> -- Use vsql's \d command to describe a table in the synced schema

=> \d hcat.messages
List of Fields by Tables
 Schema | Table | Column | Type | Size | Default | Not Null | Primary Key | Foreign
Key
-----+-----+-----+-----+-----+-----+-----+-----+-----
-----
hcat    | messages | id | int | 8 | | f | f |
hcat    | messages | userid | varchar(65000) | 65000 | | f | f |
hcat    | messages | "time" | varchar(65000) | 65000 | | f | f |
hcat    | messages | message | varchar(65000) | 65000 | | f | f |
(4 rows)

```

This example shows synchronizing with a schema created using CREATE HCATALOG SCHEMA. Synchronizing with a schema created using CREATE SCHEMA is also supported.

You can query tables in the local schema that you synchronized with an HCatalog schema. However, querying tables in a synchronized schema isn't much faster than directly querying the HCatalog schema, because SYNC_WITH_HCATALOG_SCHEMA only duplicates the HCatalog schema's metadata. The data in the table is still retrieved using the HCatalog Connector,

Data Type Conversions from Hive to Vertica

The data types recognized by Hive differ from the data types recognized by Vertica. The following table lists how the HCatalog Connector converts Hive data types into data types compatible with Vertica.

Hive Data Type	Vertica Data Type
TINYINT (1-byte)	TINYINT (8-bytes)
SMALLINT (2-bytes)	SMALLINT (8-bytes)
INT (4-bytes)	INT (8-bytes)
BIGINT (8-bytes)	BIGINT (8-bytes)
BOOLEAN	BOOLEAN
FLOAT (4-bytes)	FLOAT (8-bytes)

Hive Data Type	Vertica Data Type
DECIMAL (precision, scale)	DECIMAL (precision, scale)
DOUBLE (8-bytes)	DOUBLE PRECISION (8-bytes)
CHAR (length in characters)	CHAR (length in bytes)
VARCHAR (length in characters)	VARCHAR (length in bytes), if length <= 65000 LONG VARCHAR (length in bytes), if length > 65000
STRING (2 GB max)	VARCHAR (65000)
BINARY (2 GB max)	VARBINARY (65000)
DATE	DATE
TIMESTAMP	TIMESTAMP
LIST/ARRAY	VARCHAR (65000) containing a JSON-format representation of the list.
MAP	VARCHAR (65000) containing a JSON-format representation of the map.
STRUCT	VARCHAR (65000) containing a JSON-format representation of the struct.

Data-Width Handling Differences Between Hive and Vertica

The HCatalog Connector relies on Hive SerDe classes to extract data from files on HDFS. Therefore, the data read from these files are subject to Hive's data width restrictions. For example, suppose the SerDe parses a value for an INT column into a value that is greater than $2^{32}-1$ (the maximum value for a 32-bit integer). In this case, the value is rejected even if it would fit into a Vertica's 64-bit INTEGER column because it cannot fit into Hive's 32-bit INT.

Hive measures CHAR and VARCHAR length in characters and Vertica measures them in bytes. Therefore, if multi-byte encodings are being used (like Unicode), text might be truncated in Vertica.

Once the value has been parsed and converted to a Vertica data type, it is treated as native data. This treatment can result in some confusion when comparing the results of an identical

query run in Hive and in Vertica. For example, if your query adds two INT values that result in a value that is larger than $2^{32}-1$, the value overflows its 32-bit INT data type, causing Hive to return an error. When running the same query with the same data in Vertica using the HCatalog Connector, the value will probably still fit within Vertica's 64-int value. Thus the addition is successful and returns a value.

Using Nonstandard SerDes

Hive stores its data in unstructured flat files located in the Hadoop Distributed File System (HDFS). When you execute a Hive query, it uses a set of serializer and deserializer (SerDe) classes to extract data from these flat files and organize it into a relational database table. For Hive to be able to extract data from a file, it must have a SerDe that can parse the data the file contains. When you create a table in Hive, you can select the SerDe to be used for the table's data.

Hive has a set of standard SerDes that handle data in several formats such as delimited data and data extracted using regular expressions. You can also use third-party or custom-defined SerDes that allow Hive to process data stored in other file formats. For example, some commonly-used third-party SerDes handle data stored in JSON format.

The HCatalog Connector directly fetches file segments from HDFS and uses Hive's SerDes classes to extract data from them. The Connector includes all Hive's standard SerDes classes, so it can process data stored in any file that Hive natively supports. If you want to query data from a Hive table that uses a custom SerDe, you must first install the SerDe classes on the Vertica cluster.

Determining Which SerDe You Need

If you have access to the Hive command line, you can determine which SerDe a table uses by using Hive's SHOW CREATE TABLE statement. This statement shows the HiveQL statement needed to recreate the table. For example:

```
hive> SHOW CREATE TABLE msgjson;
OK
CREATE EXTERNAL TABLE msgjson(
  messageid int COMMENT 'from deserializer',
  userid string COMMENT 'from deserializer',
  time string COMMENT 'from deserializer',
  message string COMMENT 'from deserializer')
ROW FORMAT SERDE
'org.apache.hadoop.hive.contrib.serde2.JsonSerde'
STORED AS INPUTFORMAT
'org.apache.hadoop.mapred.TextInputFormat'
```

```
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
'hdfs://hivehost.example.com:8020/user/exampleuser/msgjson'
TBLPROPERTIES (
'transient_lastDdlTime'='1384194521')
Time taken: 0.167 seconds
```

In the example, `ROW FORMAT SERDE` indicates that a special SerDe is used to parse the data files. The next row shows that the class for the SerDe is named `org.apache.hadoop.hive.contrib.serde2.JsonSerde`. You must provide the HCatalog Connector with a copy of this SerDe class so that it can read the data from this table.

You can also find out which SerDe class you need by querying the table that uses the custom SerDe. The query will fail with an error message that contains the class name of the SerDe needed to parse the data in the table. In the following example, the portion of the error message that names the missing SerDe class is in bold.

```
=> SELECT * FROM hcat.jsontable;
ERROR 3399: Failure in UDX RPC call InvokePlanUDL(): Error in User Defined
Object [VHCatSource], error code: 0
com.vertica.sdk.UdfException: Error message is [
org.apache.hcatalog.common.HCatException : 2004 : HCatOutputFormat not
initialized, setOutput has to be called. Cause : java.io.IOException:
java.lang.RuntimeException:
MetaException(message:org.apache.hadoop.hive.serde2.SerdeException
SerDe com.cloudera.hive.serde.JSONSerde does not exist) ] HINT If error
message is not descriptive or local, may be we cannot read metadata from hive
metastore service thrift://hcathost:9083 or HDFS namenode (check
UDXLogs/UDXFencedProcessesJava.log in the catalog directory for more information)
at com.vertica.hcatalogudl.HCatalogSplitsNoOpSourceFactory
.plan(HCatalogSplitsNoOpSourceFactory.java:98)
at com.vertica.udxfence.UDXExecContext.planUDSource(UDXExecContext.java:898)
. . .
```

Installing the SerDe on the Vertica Cluster

You usually have two options to getting the SerDe class file the HCatalog Connector needs:

- Find the installation files for the SerDe, then copy those over to your Vertica cluster. For example, there are several third-party JSON SerDes available from sites like Google Code and GitHub. You may find the one that matches the file installed on your Hive cluster. If so, then download the package and copy it to your Vertica cluster.
- Directly copy the JAR files from a Hive server onto your Vertica cluster. The location for the SerDe JAR files depends on your Hive installation. On some systems, they may be located in `/usr/lib/hive/lib`.

Wherever you get the files, copy them into the `/opt/vertica/packages/hcat/lib` directory on every node in your Vertica cluster.

Important: If you add a new host to your Vertica cluster, remember to copy every custom SerDer JAR file to it.

Troubleshooting HCatalog Connector Problems

You may encounter the following issues when using the HCatalog Connector.

Connection Errors

When you use `CREATE HCATALOG SCHEMA` to create a new schema, the HCatalog Connector does not immediately attempt to connect to the HiveServer2 or metastore servers. Instead, when you execute a query using the schema or HCatalog-related system tables, the connector attempts to connect to and retrieve data from your Hadoop cluster.

The types of errors you get depend on which parameters are incorrect. Suppose you have incorrect parameters for the metastore database, but correct parameters for HiveServer2. In this case, HCatalog-related system table queries succeed, while queries on the HCatalog schema fail. The following example demonstrates creating an HCatalog schema with the correct default HiveServer2 information. However, the port number for the metastore database is incorrect.

```
=> CREATE HCATALOG SCHEMA hcat2 WITH hostname='hcahost'
-> HCATALOG_SCHEMA='default' HCATALOG_USER='hive' PORT=1234;
CREATE SCHEMA
=> SELECT * FROM HCATALOG_TABLE_LIST;
-[ RECORD 1 ]-----+-----
table_schema_id    | 45035996273864536
table_schema       | hcat2
hcatalog_schema    | default
table_name         | test
hcatalog_user_name | hive

=> SELECT * FROM hcat2.test;
ERROR 3399: Failure in UDX RPC call InvokePlanUDL(): Error in User Defined
Object [VHCatSource], error code: 0
com.vertica.sdk.UdfException: Error message is [
org.apache.hcatalog.common.HCatException : 2004 : HCatOutputFormat not
initialized, setOutput has to be called. Cause : java.io.IOException:
MetaException(message:Could not connect to meta store using any of the URIs
provided. Most recent failure: org.apache.thrift.transport.TTTransportException:
```

```
java.net.ConnectException:  
Connection refused  
at org.apache.thrift.transport.TSocket.open(TSocket.java:185)  
at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.open(  
HiveMetaStoreClient.java:277)  
... .
```

To resolve these issues, you must drop the schema and recreate it with the correct parameters. If you still have issues, determine whether there are connectivity issues between your Vertica cluster and your Hadoop cluster. Such issues can include a firewall that prevents one or more Vertica hosts from contacting the HiveServer2, metastore, or HDFS hosts.

You may also see this error if you are using HA NameNode, particularly with larger tables that HDFS splits into multiple blocks. See [Using the HCatalog Connector with HA NameNode](#) for more information about correcting this problem.

UDx Failure When Querying Data: Error 3399

You might see an error message when querying data (as opposed to metadata like schema information). This might be accompanied by a `ClassNotFoundException` in the log. This can happen for the following reasons:

- You are not using the same version of Java on your Hadoop and Vertica nodes. In this case you need to change one of them to match the other.
- You have not used `hcatUtil` to copy all Hadoop and Hive libraries and configuration files to Vertica, or you ran `hcatutil` and then changed your version of Hadoop or Hive.
- You upgraded Vertica to a new version and did not rerun `hcatutil` and reinstall the HCatalog Connector.
- The version of Hadoop you are using relies on a third-party library that you must copy manually.
- You are reading files with LZO compression and have not copied the libraries or set the `io.compression.codecs` property in `core-site.xml`.
- You are reading Parquet data from Hive, and columns were added to the table after some data was already present in the table. Adding columns does not update existing data, and the `ParquetSerDe` provided by Hive and used by the HCatalog Connector does not handle this case. This error is due to a limitation in Hive and there is no workaround.

- The query is taking too long and is timing out. If this is a frequent problem, you can increase the value of the `UDxFencedBlockTimeout` configuration parameter. See [General Parameters](#).

If you did not copy the libraries or configure LZO compression, follow the instructions in [Configuring Vertica for HCatalog](#).

If the Hive jars that you copied from Hadoop are out of date, you might see an error message like the following:

```
ERROR 3399: Failure in UDx RPC call InvokePlanUDL(): Error in User Defined Object [VHCatSource],
error code: 0 Error message is [ Found interface org.apache.hadoop.mapreduce.JobContext, but
class was expected ]
HINT hive metastore service is thrift://localhost:13433 (check UDxLogs/UDxFencedProcessesJava.log
in the catalog directory for more information)
```

This error usually signals a problem with `hive-hcatalog-core.jar`. Make sure you have an up-to-date copy of this file. Remember that if you rerun `hcatUtil` you also need to re-create the HCatalog schema.

You might also see a different form of this error:

```
ERROR 3399: Failure in UDx RPC call InvokePlanUDL(): Error in User Defined Object [VHCatSource],
error code: 0 Error message is [ javax/servlet/Filter ]
```

This error can be reported even if `hcatUtil` reports that your libraries are up to date. The `javax.servlet.Filter` class is in a library that some versions of Hadoop use but that is not usually part of the Hadoop installation directly. If you see an error mentioning this class, locate `servlet-api-*.jar` on a Hadoop node and copy it to the `hcat/lib` directory on all database nodes. If you cannot locate it on a Hadoop node, locate and download it from the Internet. (This case is rare.) The library version must be 2.3 or higher.

After you have copied the jar to the `hcat/lib` directory, reinstall the HCatalog connector as explained in [Configuring Vertica for HCatalog](#).

SerDe Errors

Errors can occur if you attempt to query a Hive table that uses a nonstandard SerDe. If you have not installed the SerDe JAR files on your Vertica cluster, you receive an error similar to the one in the following example:

```
=> SELECT * FROM hcat.jsontable;
ERROR 3399: Failure in UDx RPC call InvokePlanUDL(): Error in User Defined
Object [VHCatSource], error code: 0
com.vertica.sdk.UdfException: Error message is [
```

```
org.apache.hcatalog.common.HCatException : 2004 : HCatOutputFormat not
initialized, setOutput has to be called. Cause : java.io.IOException:
java.lang.RuntimeException:
MetaException(message:org.apache.hadoop.hive.serde2.SerDeException
SerDe com.cloudera.hive.serde.JSONSerDe does not exist) ] HINT If error
message is not descriptive or local, may be we cannot read metadata from hive
metastore service thrift://hcathost:9083 or HDFS namenode (check
UDxLogs/UDxFencedProcessesJava.log in the catalog directory for more information)
at com.vertica.hcatalogudl.HCatalogSplitsNoOpSourceFactory
.plan(HCatalogSplitsNoOpSourceFactory.java:98)
at com.vertica.udxfence.UDxExecContext.planUDSource(UDxExecContext.java:898)
. . .
```

In the error message, you can see that the root cause is a missing SerDe class (shown in bold). To resolve this issue, install the SerDe class on your Vertica cluster. See [Using Nonstandard SerDes](#) for more information.

This error may occur intermittently if just one or a few hosts in your cluster do not have the SerDe class.

Differing Results Between Hive and Vertica Queries

Sometimes, running the same query on Hive and on Vertica through the HCatalog Connector can return different results. This discrepancy is often caused by the differences between the data types supported by Hive and Vertica. See [Data Type Conversions from Hive to Vertica](#) for more information about supported data types.

If Hive string values are being truncated in Vertica, this might be caused by multi-byte character encodings in Hive. Hive reports string length in characters, while Vertica records it in bytes. For a two-byte encoding such as Unicode, you need to double the column size in Vertica to avoid truncation.

Discrepancies can also occur if the Hive table uses partition columns of types other than string.

HCatalog Connector Installation Fails on MapR

If you mount a MapR file system as an NFS mount point and then install the HCatalog Connector, it could fail with a message like the following:

```
ROLLBACK 2929: Couldn't create new UDx side process,
failed to get UDx side process info from zygote: Broken pipe
```

This might be accompanied by an error like the following in dbLog:

```
java.io.IOException: Couldn't get lock for /home/dbadmin/node02_
catalog/UDxLogs/UDxFencedProcessesJava.log
    at java.util.logging.FileHandler.openFiles(FileHandler.java:389)
    at java.util.logging.FileHandler.<init>(FileHandler.java:287)
    at com.vertica.udxfence.UDxLogger.setup(UDxLogger.java:78)
    at com.vertica.udxfence.UDxSideProcess.go(UDxSideProcess.java:75)
    ...
```

This error occurs if you locked your NFS mount point when creating it. Locking is the default. If you use the HCatalog Connector with MapR mounted as an NFS mount point, you must create the mount point with the `-o noLock` option. For example:

```
sudo mount -o noLock -t nfs MaprCLDBserviceHostname:/mapr/ClusterName/vertica/${hostname -f}/ vertica
```

You can use the HCatalog Connector with MapR without mounting the MapR file system. If you mount the MapR file system, you must do so without a lock.

Preventing Excessive Query Delays

Network issues or high system loads on the HiveServer2 server can cause long delays while querying a Hive database using the HCatalog Connector. While Vertica cannot resolve these issues, you can set parameters that limit how long Vertica waits before canceling a query on an HCatalog schema. You can set these parameters globally using Vertica configuration parameters. You can also set them for specific HCatalog schemas in the [CREATE HCATALOG SCHEMA](#) statement. These specific settings override the settings in the configuration parameters.

The `HCatConnectionTimeout` configuration parameter and the `CREATE HCATALOG SCHEMA` statement's `HCATALOG_CONNECTION_TIMEOUT` parameter control how many seconds the HCatalog Connector waits for a connection to the HiveServer2 server. A value of 0 (the default setting for the configuration parameter) means to wait indefinitely. If the server does not respond by the time this timeout elapses, the HCatalog Connector breaks the connection and cancels the query. If you find that some queries on an HCatalog schema pause excessively, try setting this parameter to a timeout value, so the query does not hang indefinitely.

The `HCatSlowTransferTime` configuration parameter and the `CREATE HCATALOG SCHEMA` statement's `HCATALOG_SLOW_TRANSFER_TIME` parameter specify how long the HCatalog Connector waits for data after making a successful connection to the server. After the specified time has elapsed, the HCatalog Connector determines whether the data transfer rate from the server is at least the value set in the `HCatSlowTransferLimit` configuration parameter (or by the `CREATE HCATALOG SCHEMA` statement's `HCATALOG_SLOW_TRANSFER_LIMIT` parameter). If it is not, then the HCatalog Connector terminates the connection and cancels the query.

You can set these parameters to cancel queries that run very slowly but do eventually complete. However, query delays are usually caused by a slow connection rather than a problem establishing the connection. Therefore, try adjusting the slow transfer rate settings first. If you find the cause of the issue is connections that never complete, you can alternately adjust the Linux TCP socket timeouts to a suitable value instead of relying solely on the `HCatConnectionTimeout` parameter.

Using HDFS Storage Locations

Vertica stores data in its native format, ROS, in storage locations. You can place storage locations on the local Linux file system or in HDFS. If you are using Premium Edition, you typically use HDFS storage locations for lower-priority data. Doing so frees space on your Vertica cluster for higher-priority data. If you are using Vertica for SQL on Apache Hadoop, you typically place ROS data only on HDFS.

If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

Requirements for HDFS Storage Locations

Caution:

If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

To store Vertica's data on HDFS, verify that:

- Your Hadoop cluster has WebHDFS enabled.
- All of the nodes in your Vertica cluster can connect to all of the nodes in your Hadoop cluster. Any firewall between the two clusters must allow connections on the ports used by HDFS. See [Testing Your Hadoop WebHDFS Configuration](#) for a procedure to test the connectivity between your Vertica and Hadoop clusters.
- If your HDFS cluster is unsecured, you have a Hadoop user whose username matches the name of the Vertica database administrator (usually named dbadmin). This Hadoop user must have read and write access to the HDFS directory where you want Vertica to store its data.
- If your HDFS cluster uses Kerberos authentication, you have a Kerberos principal for Vertica, and it has read and write access to the HDFS directory that will be used for the storage location. See [Configuring Kerberos](#). The Kerberos KDC must also be running.

- Your HDFS cluster has enough storage available for Vertica data. See [HDFS Space Requirements](#) below for details.
- The data you store in an HDFS-backed storage location does not expand your database's size beyond any data allowance in your Vertica license. Vertica counts data stored in an HDFS-backed storage location as part of any data allowance set by your license. See [Managing Licenses](#) in the Administrator's Guide for more information.

HDFS Space Requirements

If your Vertica database is K-safe, HDFS-based storage locations contain two copies of the data you store in them. One copy is the primary projection, and the other is the buddy projection. If you have enabled HDFS's data-redundancy feature, Hadoop stores both projections multiple times. This duplication might seem excessive. However, it is similar to how a RAID level 1 or higher stores redundant copies of both the primary and buddy projections. The redundant copies also help the performance of HDFS by enabling multiple nodes to process a request for a file.

Verify that your HDFS installation has sufficient space available for redundant storage of both the primary and buddy projections of your K-safe data. You can adjust the number of duplicates stored by HDFS by setting the `HadoopFSReplication` configuration parameter. See [Troubleshooting HDFS Storage Locations](#) for details.

Additional Requirements for Backing Up Data Stored on HDFS

To back up your data stored in HDFS storage locations, your Hadoop cluster must have snapshotting enabled for the directories to be used for backups. The easiest way to do this is to give the database administrator's account superuser privileges in Hadoop, so that snapshotting can be set automatically. Alternatively, use Hadoop to enable snapshotting for each directory before using it for backups.

In addition, your Vertica database must:

- Have enough Hadoop components and libraries installed to run the `Hadoop distcp` command as the Vertica database-administrator user (usually `dbadmin`).
- Have the `JavaBinaryForUDx` and `HadoopHome` configuration parameters set correctly.

Caution: After you have created an HDFS storage location, full database backups will fail with the error message:

```
ERROR 5127: Unable to create snapshot No such file /usr/bin/hadoop:  
check the HadoopHome configuration parameter
```

This error is caused by the backup script not being able to back up the HDFS storage locations. You must configure Vertica and Hadoop to enable the backup script to back these locations. After you configure Vertica and Hadoop, you can once again perform full database backups.

See [Backing Up HDFS Storage Locations](#) for details on configuring your Vertica and Hadoop clusters to enable HDFS storage location backup.

Best Practices for SQL on Apache Hadoop

If you are using the Vertica for SQL on Apache Hadoop product, Micro Focus recommends the following best practices for storage locations:

- Place only data type storage locations on HDFS storage.
- Place temp space directly on the local Linux file system, not in HDFS.
- For the best performance, place the Vertica catalog directly on the local Linux file system.
- Create the database first on a local Linux file system. Then, you can extend the database to HDFS storage locations and set storage policies that exclusively place data blocks on the HDFS storage location.
- For better performance, if you are running Vertica only on a subset of the HDFS nodes, do not run the HDFS balancer on them. The HDFS balancer can move data blocks farther away, causing Vertica to read non-local data during query execution. Queries run faster if they do not require network I/O.

Generally, HDFS requires approximately 2 GB of memory for each node in the cluster. To support this requirement in your Vertica configuration:

1. Create a 2-GB resource pool.
2. Do not assign any Vertica execution resources to this pool. This approach reserves the space for use by HDFS.

Alternatively, use Ambari or Cloudera Manager to find the maximum heap size required by HDFS and set the size of the resource pool to that value.

For more about how to configure resource pools, see [Managing Workloads](#).

How the HDFS Storage Location Stores Data

Vertica stores data in storage locations on HDFS similarly to the way it stores data in the Linux file system. See [Managing Storage Locations](#) in the Administrator's Guide for more information about storage locations. When you create a storage location on HDFS, Vertica stores the ROS containers holding its data on HDFS. You can choose which data uses the HDFS storage location: from the data for just a single table or partition to all of the database's data.

When Vertica reads data from or writes data to an HDFS storage location, the node storing or retrieving the data contacts the Hadoop cluster directly to transfer the data. If a single ROS container file is split among several HDFS nodes, the Vertica node connects to each of them. The Vertica node retrieves the pieces and reassembles the file. Because each node fetches its own data directly from the source, data transfers are parallel, increasing their efficiency. Having the Vertica nodes directly retrieve the file splits also reduces the impact on the Hadoop cluster.

What You Can Store in HDFS

Use HDFS storage locations to store only data. You cannot store catalog information in an HDFS storage location.

Caution: While it is possible to use an HDFS storage location for temporary data storage, you must never do so. Using HDFS for temporary storage causes severe performance issues.

What HDFS Storage Locations Cannot Do

Because Vertica uses storage locations to store ROS containers in a proprietary format, MapReduce and other Hadoop components cannot access your Vertica ROS data stored in HDFS. Never allow another program that has access to HDFS to write to the ROS files. Any outside modification of these files can lead to data corruption and loss. Applications must use the [Vertica client libraries](#) to access Vertica data. If you want to share ROS data with other Hadoop components, you can export it (see [Exporting Data](#)).

The storage location stores and reads only ROS containers. It cannot read data stored in native formats in HDFS. See [Hadoop Interfaces](#) for other ways to read data stored in HDFS.

Creating an HDFS Storage Location

Use the [CREATE LOCATION](#) statement to create an HDFS storage location. Make the following changes from creating local storage locations:

- For the path, use the WebHDFS URI for the HDFS directory where you want Vertica to store the location's data. This URI is the same as a standard HDFS URL, except it uses the `webhdfs://` protocol and its path does not start with `/webhdfs/v1/`.
- Include the `ALL NODES SHARED` keywords, as all HDFS storage locations are shared storage. This is required even if you have only one HDFS node in your cluster.

Caution:

If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

Creating the Storage Location

To create an HDFS storage location, first create the location on all nodes and then set its storage policy to HDFS. To create the location in HDFS on all nodes:

```
=> CREATE LOCATION 'webhdfs://hadoop:50070/user/dbadmin' ALL NODES SHARED  
    USAGE 'data' LABEL 'coldstorage';
```

Next, set the storage policy for your database objects to use this location:

```
=> SELECT SET_OBJECT_STORAGE_POLICY('SchemaName', 'coldstorage');
```

This causes all data in the named schema to be written to the HDFS storage location (`coldstorage`) instead of the local disk. You can set storage policies for a schema, a table, a partition, or the entire database.

For more information, see [Managing Storage Locations](#).

Adding HDFS Storage Locations to New Nodes

If you add nodes to your Vertica cluster, they do not automatically have access to existing HDFS storage locations. You must manually create the storage location for the new node using the `CREATE LOCATION` statement. Do not use the `ALL NODES` keyword in this statement. Instead, use the `NODE` keyword with the name of the new node to tell Vertica that just that node needs to add the shared location.

Caution: You must manually create the storage location. Otherwise, the new node uses the default storage policy (usually, storage on the local Linux filesystem) to store data that the other nodes store in HDFS. As a result, the node can run out of disk space.

The following example shows how to add the storage location from the preceding example to a new node named `v_vmart_node0004`:

```
=> CREATE LOCATION 'webhdfs://hadoop:50070/user/dbadmin' NODE 'v_vmart_node0004'  
    SHARED USAGE 'data' LABEL 'coldstorage';
```

Any [active standby nodes](#) in your cluster when you create an HDFS-based storage location automatically create their own instances of the location. When the standby node takes over for a down node, it uses its own instance of the location to store data for objects using the HDFS-based storage policy. Treat standby nodes added after you create the storage location as any other new node. You must manually define the HDFS storage location.

Creating a Storage Policy for HDFS Storage Locations

After you create an HDFS storage location, you assign database objects to the location by setting storage policies. Based on these storage policies, database objects such as partition ranges, individual tables, whole schemas, or even the entire database store their data in the HDFS storage location. Use the [SET_OBJECT_STORAGE_POLICY](#) function to assign objects to an HDFS storage location. In the function call, supply the label you assigned to the HDFS storage location as the location label argument. You do so using the `CREATE LOCATION` statement's `LABEL` keyword.

The following example demonstrates using [SET_OBJECT_STORAGE_POLICY](#) to store a table in an HDFS storage location. The example statement sets the policy for an existing table, named `messages`, to store its data in an HDFS storage location, named `coldstorage`.

```
=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'coldstorage');
```

This table's data is moved to the HDFS storage location with the next merge-out. Alternatively, you can have Vertica move the data immediately by using the `enforce_storage_move` parameter.

You can query the [STORAGE_CONTAINERS](#) system table and examine the `location_label` column to verify that Vertica has moved the data:

```
=> SELECT node_name, projection_name, location_label, total_row_count FROM V_MONITOR.STORAGE_
CONTAINERS
WHERE projection_name ILIKE 'messages%';
 node_name      | projection_name | location_label | total_row_count
-----+-----+-----+-----
v_vmart_node0001 | messages_b0    | coldstorage   | 366057
v_vmart_node0001 | messages_b1    | coldstorage   | 366511
v_vmart_node0002 | messages_b0    | coldstorage   | 367432
v_vmart_node0002 | messages_b1    | coldstorage   | 366057
v_vmart_node0003 | messages_b0    | coldstorage   | 366511
v_vmart_node0003 | messages_b1    | coldstorage   | 367432
(6 rows)
```

See [Creating Storage Policies](#) in the Administrator's Guide for more information about assigning storage policies to objects.

Backing Up HDFS Storage Locations

Micro Focus recommends that you regularly back up the data in your Vertica database. This recommendation includes data stored in your HDFS storage locations. The Vertica backup script (`vbr`) can back up HDFS storage locations. However, you must perform several configuration steps before it can back up these locations.

Caution: After you have created an HDFS storage location, full database backups will fail with the error message:

```
ERROR 5127: Unable to create snapshot No such file /usr/bin/hadoop:
check the HadoopHome configuration parameter
```

This error is caused by the backup script not being able to back up the HDFS storage locations. You must configure Vertica and Hadoop to enable the backup script to back these locations. After you configure Vertica and Hadoop, you can once again perform full database backups.

There are several considerations for backing up HDFS storage locations in your database:

- HDFS storage locations do not support object-level backups. You must perform a full database backup to back up the data in your HDFS storage locations.
- Data in an HDFS storage location is backed up to HDFS. This backup guards against accidental deletion or corruption of data. It does not prevent data loss in the case of a catastrophic failure of the entire Hadoop cluster. To prevent data loss, you must have a backup and disaster recovery plan for your Hadoop cluster.

Data stored on the Linux native filesystem is still backed up to the location you specify in the backup configuration file. It and the data in HDFS storage locations are handled separately by the vbr backup script.

- You must configure your Vertica cluster to restore database backups containing an HDFS storage location. See [Configuring Vertica to Restore HDFS Storage Locations](#) for the configuration steps you must take.
- The HDFS directory for the storage location must have snapshotting enabled. You can either directly configure this yourself or enable the database administrator's Hadoop account to do it for you automatically. See [Configuring Hadoop and Vertica to Enable Backup of HDFS Storage](#) for more information.

The topics in this section explain the configuration steps you must take to enable the backup of HDFS storage locations.

Configuring Hadoop and Vertica to Enable Backup of HDFS Storage

The Vertica backup script uses HDFS's snapshotting feature to create a backup of HDFS storage locations. A directory must allow snapshotting before HDFS can take a snapshot. Only a Hadoop superuser can enable snapshotting on a directory. Vertica can enable snapshotting automatically if the database administrator is also a Hadoop superuser.

If HDFS is unsecured, the following instructions apply to the database administrator account, usually dbadmin. If HDFS uses Kerberos security, the following instructions apply to the principal stored in the Vertica keytab file, usually vertica. The instructions below use the term "database account" to refer to this user.

We recommend that you make the database administrator or principal a Hadoop superuser. If you are not able to do so, you must enable snapshotting on the directory before configuring it for use by Vertica.

The steps you need to take to make the Vertica database administrator account a superuser depend on the distribution of Hadoop you are using. Consult your Hadoop distribution's documentation for details.

Manually Enabling Snapshotting for a Directory

If you cannot grant superuser status to the database account, you can instead enable snapshotting of each directory manually. Use the following command:

```
hdfs dfsadmin -allowSnapshot path
```

Issue this command for each directory on each node. Remember to do this each time you add a new node to your HDFS cluster.

Nested snapshottable directories are not allowed, so you cannot enable snapshotting for a parent directory to automatically enable it for child directories. You must enable it for each individual directory.

Additional Requirements for Kerberos

If HDFS uses Kerberos, then in addition to granting the keytab principal access, you must set a Vertica configuration parameter. In Vertica, set the `HadoopConfDir` parameter to the location of the directory containing the `core-site.xml`, `hdfs-site.xml`, and `yarn-site.xml` configuration files:

```
=> ALTER DATABASE example SET HadoopConfDir = '/etc/hadoop/conf';
```

All three configuration files must be present in this directory.

If your Vertica nodes are not co-located on HDFS nodes, then you must copy these files from an HDFS node to each Vertica node. Use the same path on every database node, because `HadoopConfDir` is a global value.

Testing the Database Account's Ability to Make HDFS Directories Snapshottable

After making the database account a Hadoop superuser, verify that the account can set directories snapshottable:

1. Log into the Hadoop cluster as the database account (dbadmin by default).
2. Determine a location in HDFS where the database administrator can create a directory. The /tmp directory is usually available. Create a test HDFS directory using the command:

```
$ hdfs dfs -mkdir /path/testdir
```

3. Make the test directory snapshottable using the command:

```
$ hdfs dfsadmin -allowSnapshot /path/testdir
```

The following example demonstrates creating an HDFS directory and making it snapshottable:

```
$ hdfs dfs -mkdir /tmp/snaptest  
$ hdfs dfsadmin -allowSnapshot /tmp/snaptest  
Allowing snapshot on /tmp/snaptest succeeded
```

Configuring Vertica to Restore HDFS Storage Locations

Your Vertica cluster must be able to run the Hadoop distcp command to restore a backup of an HDFS storage location. The easiest way to enable your cluster to run this command is to install several Hadoop packages on each node. These packages must be from the same distribution and version of Hadoop that is running on your Hadoop cluster.

The steps you need to take depend on:

- The distribution and version of Hadoop running on the Hadoop cluster containing your HDFS storage location.
- The distribution of Linux running on your Vertica cluster.

Note: Installing the Hadoop packages necessary to run distcp does not turn your Vertica database into a Hadoop cluster. This process installs just enough of the Hadoop support files on your cluster to run the distcp command. There is no additional overhead placed on the Vertica cluster, aside from a small amount of additional disk space consumed by the Hadoop support files.

Configuration Overview

The steps for configuring your Vertica cluster to restore backups for HDFS storage location are:

1. If necessary, install and configure a Java runtime on the hosts in the Vertica cluster.
2. Find the location of your Hadoop distribution's package repository.
3. Add the Hadoop distribution's package repository to the Linux package manager on all hosts in your cluster.
4. Install the necessary Hadoop packages on your Vertica hosts.
5. Set two configuration parameters in your Vertica database related to Java and Hadoop.
6. If your HDFS storage location uses Kerberos, set additional configuration parameters to allow Vertica user credentials to be proxied.
7. Confirm that the Hadoop distcp command runs on your Vertica hosts.

The following sections describe these steps in greater detail.

Installing a Java Runtime

You Vertica cluster must have a Java Virtual Machine (JVM) installed to run the Hadoop distcp command. It already has a JVM installed if you have configured it to:

- Execute User-Defined Extensions developed in Java. See [Developing User-Defined Extensions \(UDxs\)](#) for more information.
- Access Hadoop data using the HCatalog Connector. See [Using the HCatalog Connector](#) for more information.

If your Vertica database has a JVM installed, verify that your Hadoop distribution supports it. See your Hadoop distribution's documentation to determine which JVMs it supports.

If the JVM installed on your Vertica cluster is not supported by your Hadoop distribution you must uninstall it. Then you must install a JVM that is supported by both Vertica and your Hadoop distribution. See [Vertica SDKs](#) in Supported Platforms for a list of the JVMs compatible with Vertica.

If your Vertica cluster does not have a JVM (or its existing JVM is incompatible with your Hadoop distribution), follow the instructions in [Installing the Java Runtime on Your Vertica Cluster](#).

Finding Your Hadoop Distribution's Package Repository

Many Hadoop distributions have their own installation system, such as Cloudera's Manager or Hortonwork's Ambari. However, they also support manual installation using native Linux packages such as RPM and .deb files. These package files are maintained in a repository. You can configure your Vertica hosts to access this repository to download and install Hadoop packages.

Consult your Hadoop distribution's documentation to find the location of its Linux package repository. This information is often located in the portion of the documentation covering manual installation techniques.:

Each Hadoop distribution maintains separate repositories for each of the major Linux package management systems. Find the specific repository for the Linux distribution running on your Vertica cluster. Be sure that the package repository that you select matches the version of Hadoop distribution installed on your Hadoop cluster.

Configuring Vertica Nodes to Access the Hadoop Distribution's Package Repository

Configure the nodes in your Vertica cluster so they can access your Hadoop distribution's package repository. Your Hadoop distribution's documentation should explain how to add the repositories to your Linux platform. If the documentation does not explain how to add the repository to your packaging system, refer to your Linux distribution's documentation.

The steps you need to take depend on the package management system your Linux platform uses. Usually, the process involves:

- Downloading a configuration file.
- Adding the configuration file to the package management system's configuration directory.
- For Debian-based Linux distributions, adding the Hadoop repository encryption key to the root account keyring.
- Updating the package management system's index to have it discover new packages.

The following example demonstrates adding the Hortonworks 2.1 package repository to an Ubuntu 12.04 host. These steps in this example are explained in the Hortonworks documentation.

```
$ wget http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/hdp.list \  
-O /etc/apt/sources.list.d/hdp.list  
--2014-08-20 11:06:00-- http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/hdp.list  
Connecting to 16.113.84.10:8080... connected.  
Proxy request sent, awaiting response... 200 OK  
Length: 161 [binary/octet-stream]  
Saving to: `/etc/apt/sources.list.d/hdp.list'  
  
100%[=====>] 161          --.-K/s   in 0s  
  
2014-08-20 11:06:00 (8.00 MB/s) - `/etc/apt/sources.list.d/hdp.list' saved [161/161]  
  
$ gpg --keyserver pgp.mit.edu --recv-keys B9733A7A07513CAD  
gpg: requesting key 07513CAD from hkp server pgp.mit.edu  
gpg: /root/.gnupg/trustdb.gpg: trustdb created  
gpg: key 07513CAD: public key "Jenkins (HDP Builds) <jenkin@hortonworks.com>" imported  
gpg: Total number processed: 1  
gpg:          imported: 1 (RSA: 1)  
  
$ gpg -a --export 07513CAD | apt-key add -  
OK  
  
$ apt-get update  
Hit http://us.archive.ubuntu.com precise Release.gpg  
Hit http://extras.ubuntu.com precise Release.gpg  
Get:1 http://security.ubuntu.com precise-security Release.gpg [198 B]  
Hit http://us.archive.ubuntu.com precise-updates Release.gpg  
Get:2 http://public-repo-1.hortonworks.com HDP-UTILS Release.gpg [836 B]  
Get:3 http://public-repo-1.hortonworks.com HDP Release.gpg [836 B]  
Hit http://us.archive.ubuntu.com precise-backports Release.gpg  
Hit http://extras.ubuntu.com precise Release  
Get:4 http://security.ubuntu.com precise-security Release [50.7 kB]  
Get:5 http://public-repo-1.hortonworks.com HDP-UTILS Release [6,550 B]  
Hit http://us.archive.ubuntu.com precise Release  
Hit http://extras.ubuntu.com precise/main Sources  
Get:6 http://public-repo-1.hortonworks.com HDP Release [6,502 B]  
Hit http://us.archive.ubuntu.com precise-updates Release  
Get:7 http://public-repo-1.hortonworks.com HDP-UTILS/main amd64 Packages [1,955 B]  
Get:8 http://security.ubuntu.com precise-security/main Sources [108 kB]  
Get:9 http://public-repo-1.hortonworks.com HDP-UTILS/main i386 Packages [762 B]  
.  
.  
.  
Reading package lists... Done
```

You must add the Hadoop repository to all hosts in your Vertica cluster.

Installing the Required Hadoop Packages

After configuring the repository, you are ready to install the Hadoop packages. The packages you need to install are:

- hadoop
- hadoop-hdfs

- `hadoop-client`

The names of the packages are usually the same across all Hadoop and Linux distributions. These packages often have additional dependencies. Always accept any additional packages that the Linux package manager asks to install.

To install these packages, use the package manager command for your Linux distribution. The package manager command you need to use depends on your Linux distribution:

- On Red Hat and CentOS, the package manager command is `yum`.
- On Debian and Ubuntu, the package manager command is `apt-get`.
- On SUSE the package manager command is `zypper`.

Consult your Linux distribution's documentation for instructions on installing packages.

The following example demonstrates installing the required Hadoop packages from the Hortonworks 2.1 distribution on an Ubuntu 12.04 system.

```
# apt-get install hadoop hadoop-hdfs hadoop-client
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bigtop-jsvc hadoop-mapreduce hadoop-yarn zookeeper
The following NEW packages will be installed:
  bigtop-jsvc hadoop hadoop-client hadoop-hdfs hadoop-mapreduce hadoop-yarn
  zookeeper
0 upgraded, 7 newly installed, 0 to remove and 90 not upgraded.
Need to get 86.6 MB of archives.
After this operation, 99.8 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  bigtop-jsvc amd64 1.0.10-1 [28.5 kB]
Get:2 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  zookeeper all 3.4.5.2.1.3.0-563 [6,820 kB]
Get:3 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop all 2.4.0.2.1.3.0-563 [21.5 MB]
Get:4 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-hdfs all 2.4.0.2.1.3.0-563 [16.0 MB]
Get:5 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-yarn all 2.4.0.2.1.3.0-563 [15.1 MB]
Get:6 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-mapreduce all 2.4.0.2.1.3.0-563 [27.2 MB]
Get:7 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-client all 2.4.0.2.1.3.0-563 [3,650 B]
Fetched 86.6 MB in 1min 2s (1,396 kB/s)
Selecting previously unselected package bigtop-jsvc.
(Reading database ... 197894 files and directories currently installed.)
Unpacking bigtop-jsvc (from ../bigtop-jsvc_1.0.10-1_amd64.deb) ...
Selecting previously unselected package zookeeper.
Unpacking zookeeper (from ../zookeeper_3.4.5.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop.
```

```
Unpacking hadoop (from ../hadoop_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-hdfs.
Unpacking hadoop-hdfs (from ../hadoop-hdfs_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-yarn.
Unpacking hadoop-yarn (from ../hadoop-yarn_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-mapreduce.
Unpacking hadoop-mapreduce (from ../hadoop-mapreduce_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-client.
Unpacking hadoop-client (from ../hadoop-client_2.4.0.2.1.3.0-563_all.deb) ...
Processing triggers for man-db ...
Setting up bigtop-jsvc (1.0.10-1) ...
Setting up zookeeper (3.4.5.2.1.3.0-563) ...
update-alternatives: using /etc/zookeeper/conf.dist to provide /etc/zookeeper/conf (zookeeper-conf)
in auto mode.
Setting up hadoop (2.4.0.2.1.3.0-563) ...
update-alternatives: using /etc/hadoop/conf.empty to provide /etc/hadoop/conf (hadoop-conf) in auto
mode.
Setting up hadoop-hdfs (2.4.0.2.1.3.0-563) ...
Setting up hadoop-yarn (2.4.0.2.1.3.0-563) ...
Setting up hadoop-mapreduce (2.4.0.2.1.3.0-563) ...
Setting up hadoop-client (2.4.0.2.1.3.0-563) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
```

Setting Configuration Parameters

You must set two configuration parameters to enable Vertica to restore HDFS data:

- `JavaBinaryForUDx` is the path to the Java executable. You may have already set this value to use Java UDxs or the HCatalog Connector. You can find the path for the default Java executable from the Bash command shell using the command:

```
which java
```

- `HadoopHome` is the path where Hadoop is installed on the Vertica hosts. This is the directory that contains `bin/hadoop` (the bin directory containing the Hadoop executable file). The default value for this parameter is `/usr`. The default value is correct if your Hadoop executable is located at `/usr/bin/hadoop`.

The following example demonstrates setting and then reviewing the values of these parameters.

```
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
=> SELECT get_config_parameter('JavaBinaryForUDx');
   get_config_parameter
-----
/usr/bin/java
(1 row)
=> ALTER DATABASE mydb SET HadoopHome = '/usr';
```

```
=> SELECT get_config_parameter('HadoopHome');
       get_config_parameter
-----
      /usr
(1 row)
```

There are additional parameters you may, optionally, set:

- `HadoopFSReadRetryTimeout` and `HadoopFSWriteRetryTimeout` specify how long to wait before failing. The default value for each is 180 seconds. If you are confident that your file system will fail more quickly, you can potentially improve performance by lowering these values.
- `HadoopFSReplication` is the number of replicas HDFS makes. By default the Hadoop client chooses this; Vertica uses the same value for all nodes. We recommend against changing this unless directed to.
- `HadoopFSBlockSizeBytes` is the block size to write to HDFS; larger files are divided into blocks of this size. The default is 64MB.

Setting Kerberos Parameters

If your Vertica nodes are co-located on HDFS nodes and you are using Kerberos, you must change some Hadoop configuration parameters. These changes are needed in order for restoring from backups to work. In `yarn-site.xml` on every Vertica node, set the following parameters:

Parameter	Value
<code>yarn.resourcemanager.proxy-user-privileges.enabled</code>	true
<code>yarn.resourcemanager.proxyusers.*.groups</code>	*
<code>yarn.resourcemanager.proxyusers.*.hosts</code>	*
<code>yarn.resourcemanager.proxyusers.*.users</code>	*
<code>yarn.timeline-service.http-authentication.proxyusers.*.groups</code>	*
<code>yarn.timeline-service.http-authentication.proxyusers.*.hosts</code>	*
<code>yarn.timeline-service.http-authentication.proxyusers.*.users</code>	*

No changes are needed on HDFS nodes that are not also Vertica nodes.

Confirming that distcp Runs

After the packages are installed on all hosts in your cluster, your database should be able to run the Hadoop distcp command. To test it:

1. Log into any host in your cluster as the database administrator.
2. At the Bash shell, enter the command:

```
$ hadoop distcp
```

3. The command should print a message similar to the following:

```
usage: distcp OPTIONS [source_path...] <target_path>
       OPTIONS
  -async                Should distcp execution be blocking
  -atomic               Commit all changes or none
  -bandwidth <arg>     Specify bandwidth per map in MB
  -delete               Delete from target, files missing in source
  -f <arg>              List of files that need to be copied
  -filelimit <arg>     (Deprecated!) Limit number of files copied to <= n
  -i                    Ignore failures during copy
  -log <arg>           Folder on DFS where distcp execution logs are
                       saved
  -m <arg>              Max number of concurrent maps to use for copy
  -mapredSslConf <arg> Configuration for ssl config file, to use with
                       https://
  -overwrite            Choose to overwrite target files unconditionally,
                       even if they exist.
  -p <arg>              preserve status (rbugpc)(replication, block-size,
                       user, group, permission, checksum-type)
  -sizelimit <arg>     (Deprecated!) Limit number of files copied to <= n
                       bytes
  -skipcrccheck         Whether to skip CRC checks between source and
                       target paths.
  -strategy <arg>      Copy strategy to use. Default is dividing work
                       based on file sizes
  -tmp <arg>            Intermediate work path to be used for atomic
                       commit
  -update               Update target, copying only missingfiles or
                       directories
```

4. Repeat these steps on the other hosts in your database to verify that all of the hosts can run distcp.

Troubleshooting

If you cannot run the distcp command, try the following steps:

- If Bash cannot find the `hadoop` command, you may need to manually add Hadoop's `bin` directory to the system search path. An alternative is to create a symbolic link in an existing directory in the search path (such as `/usr/bin`) to the `hadoop` binary.
- Ensure the version of Java installed on your Vertica cluster is compatible with your Hadoop distribution.
- Review the Linux package installation tool's logs for errors. In some cases, packages may not be fully installed, or may not have been downloaded due to network issues.
- Ensure that the database administrator account has permission to execute the `hadoop` command. You may need to add the account to a specific group in order to allow it to run the necessary commands.

Performing Backups Containing HDFS Storage Locations

After you configure Hadoop and Vertica, HDFS storage locations are automatically backed up when you perform a full database backup. If you already have a backup configuration file for a full database backup, you do not need to make any changes to it. You just run the `vbr` backup script as usual to perform the full database backup. See [Creating Full Backups](#) in the Administrator's Guide for instructions on running the `vbr` backup script.

If you do not have a backup configuration file for a full database backup, you must create one to back up the data in your HDFS storage locations. See [Creating vbr Configuration Files](#) in the Administrator's Guide for more information.

Removing HDFS Storage Locations

The steps to remove an HDFS storage location are similar to standard storage locations:

1. Remove any existing data from the HDFS storage location by using [SET_OBJECT_STORAGE_POLICY](#) to change each object's storage location. Alternatively, you can use [CLEAR_OBJECT_STORAGE_POLICY](#). Because the Tuple Mover runs infrequently, set the `enforce_storage_move` parameter to `true` to make the change immediately.
2. Retire the location on each host that has the storage location defined by using [RETIRE_LOCATION](#). Set `enforce_storage_move` to `true`.

3. Drop the location on each host that has the storage location defined by using [DROP_LOCATION](#).
4. Optionally remove the snapshots and files from the HDFS directory for the storage location.

For more information about changing storage policies, changing usage, retiring locations, and dropping locations, see [Managing Storage Locations](#) in the Administrator's Guide.

Important: If you have backed up the data in the HDFS storage location you are removing, you must perform a full database backup after you remove the location. If you do not and restore the database to a backup made before you removed the location, the location's data is restored.

Removing Storage Location Files from HDFS

Dropping an HDFS storage location does not automatically clean the HDFS directory that stored the location's files. Any snapshots of the data files created when backing up the location are also not deleted. These files consume disk space on HDFS and also prevent the directory from being reused as an HDFS storage location. Vertica refuses to create a storage location in a directory that contains existing files or subdirectories. You must log into the Hadoop cluster to delete the files from HDFS. An alternative is to use some other HDFS file management tool.

Removing Backup Snapshots

HDFS returns an error if you attempt to remove a directory that has snapshots:

```
$ hdfs dfs -rm -r -f -skipTrash /user/dbadmin/v_vmart_node0001
rm: The directory /user/dbadmin/v_vmart_node0001 cannot be deleted since
/user/dbadmin/v_vmart_node0001 is snapshottable and already has snapshots
```

The Vertica backup script creates snapshots of HDFS storage locations as part of the backup process. See [Backing Up HDFS Storage Locations](#) for more information. If you made backups of your HDFS storage location, you must delete the snapshots before removing the directories.

HDFS stores snapshots in a subdirectory named `.snapshot`. You list the snapshots in the directory using the standard HDFS `ls` command. The following example demonstrates listing the snapshots defined for node0001.

```
$ hdfs dfs -ls /user/dbadmin/v_vmart_node0001/.snapshot
Found 1 items
```

```
drwxrwx--- - dbadmin supergroup          0 2014-09-02 10:13 /user/dbadmin/v_vmart_
node0001/.snapshot/s20140902-101358.629
```

To remove snapshots, use the command:

```
hdfs dfs -removeSnapshot directory snapshotname
```

The following example demonstrates the command to delete the snapshot shown in the previous example:

```
$ hdfs dfs -deleteSnapshot /user/dbadmin/v_vmart_node0001 s20140902-101358.629
```

You must delete each snapshot from the directory for each host in the cluster. After you have deleted the snapshots, you can delete the directories in the storage location.

Important: Each snapshot's name is based on a timestamp down to the millisecond. Nodes independently create their own snapshot. They do not synchronize snapshot creation, so their snapshot names differ. You must list each node's snapshot directory to learn the names of the snapshots it contains.

See Apache's [HDFS Snapshot documentation](#) for more information about managing and removing snapshots.

Removing the Storage Location Directories

You can remove the directories that held the storage location's data by either of the following methods:

- Use an HDFS file manager to delete directories. See your Hadoop distribution's documentation to determine if it provides a file manager.
- Log into the Hadoop NameNode using the database administrator's account and use HDFS's `rmr` command to delete the directories. See Apache's [File System Shell Guide](#) for more information.

The following example uses the HDFS `rmr` command from the Linux command line to delete the directories left behind in the HDFS storage location directory `/user/dbadmin`. It uses the `-skipTrash` flag to force the immediate deletion of the files.

```
$ hdfs dfs -ls /user/dbadmin
Found 3 items
drwxrwx--- - dbadmin supergroup          0 2014-08-29 15:11 /user/dbadmin/v_vmart_node0001
drwxrwx--- - dbadmin supergroup          0 2014-08-29 15:11 /user/dbadmin/v_vmart_node0002
drwxrwx--- - dbadmin supergroup          0 2014-08-29 15:11 /user/dbadmin/v_vmart_node0003

$ hdfs dfs -rmr -skipTrash /user/dbadmin/*
```

```
Deleted /user/dbadmin/v_vmart_node0001  
Deleted /user/dbadmin/v_vmart_node0002  
Deleted /user/dbadmin/v_vmart_node0003
```

Troubleshooting HDFS Storage Locations

This topic explains some common issues with HDFS storage locations.

HDFS Storage Disk Consumption

By default, HDFS makes three copies of each file it stores. This replication helps prevent data loss due to disk or system failure. It also helps increase performance by allowing several nodes to handle a request for a file.

A Vertica database with a K-Safety value of 1 or greater also stores its data redundantly using buddy projections.

When a K-Safe Vertica database stores data in an HDFS storage location, its data redundancy is compounded by HDFS's redundancy. HDFS stores three copies of the primary projection's data, plus three copies of the buddy projection for a total of six copies of the data.

If you want to reduce the amount of disk storage used by HDFS locations, you can alter the number of copies of data that HDFS stores. The Vertica configuration parameter named `HadoopFSReplication` controls the number of copies of data HDFS stores.

You can determine the current HDFS disk usage by logging into the Hadoop NameNode and issuing the command:

```
hdfs dfsadmin -report
```

This command prints the usage for the entire HDFS storage, followed by details for each node in the Hadoop cluster. The following example shows the beginning of the output from this command, with the total disk space highlighted:

```
$ hdfs dfsadmin -report  
Configured Capacity: 51495516981 (47.96 GB)  
Present Capacity: 32087212032 (29.88 GB)  
DFS Remaining: 31565144064 (29.40 GB)  
DFS Used: 522067968 (497.88 MB)  
DFS Used%: 1.63%  
Under replicated blocks: 0  
Blocks with corrupt replicas: 0  
Missing blocks: 0  
...
```

After loading a simple million-row table into a table stored in an HDFS storage location, the report shows greater disk usage:

```
Configured Capacity: 51495516981 (47.96 GB)
Present Capacity: 32085299338 (29.88 GB)
DFS Remaining: 31373565952 (29.22 GB)
DFS Used: 711733386 (678.76 MB)
DFS Used%: 2.22%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
. . .
```

The following Vertica example demonstrates:

1. Dropping the table in Vertica.
2. Setting the HadoopFSReplication configuration option to 1. This tells HDFS to store a single copy of an HDFS storage location's data.
3. Recreating the table and reloading its data.

```
=> DROP TABLE messages;
DROP TABLE

=> ALTER DATABASE mydb SET HadoopFSReplication = 1;

=> CREATE TABLE messages (id INTEGER, text VARCHAR);
CREATE TABLE

=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'hdfs');
SET_OBJECT_STORAGE_POLICY
-----
Object storage policy set.
(1 row)

=> COPY messages FROM '/home/dbadmin/messages.txt' DIRECT;
Rows Loaded
-----
1000000
```

Running the HDFS report on Hadoop now shows less disk space use:

```
$ hdfs dfsadmin -report
Configured Capacity: 51495516981 (47.96 GB)
Present Capacity: 32086278190 (29.88 GB)
DFS Remaining: 31500988416 (29.34 GB)
DFS Used: 585289774 (558.18 MB)
DFS Used%: 1.82%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
. . .
```

Caution: Reducing the number of copies of data stored by HDFS increases the risk of data loss. It can also negatively impact the performance of HDFS by reducing the number of nodes that can provide access to a file. This slower performance can impact the performance of Vertica queries that involve data stored in an HDFS storage location.

Kerberos Authentication When Creating a Storage Location

If HDFS uses Kerberos authentication, then the CREATE LOCATION statement authenticates using the Vertica keytab principal, not the principal of the user performing the action. If the creation fails with an authentication error, verify that you have followed the steps described in [Configuring Kerberos](#) to configure this principal.

When creating an HDFS storage location on a Hadoop cluster using Kerberos, CREATE LOCATION reports the principal being used as in the following example:

```
=> CREATE LOCATION 'webhdfs://hadoop.example.com:50070/user/dbadmin' ALL NODES SHARED
      USAGE 'data' LABEL 'coldstorage';
NOTICE 0: Performing HDFS operations using kerberos principal [vertica/hadoop.example.com]
CREATE LOCATION
```

Backup or Restore Fails When Using Kerberos

When backing up an HDFS storage location that uses Kerberos, you might see an error such as:

```
createSnapshot: Failed on local exception: java.io.IOException:
java.lang.IllegalArgumentException: Server has invalid Kerberos principal:
hdfs/test.example.com@EXAMPLE.COM;
```

When restoring an HDFS storage location that uses Kerberos, you might see an error such as:

```
Error msg: Initialization thread logged exception:
Distcp failure!
```

Either of these failures means that Vertica could not find the required configuration files in the HadoopConfDir directory. Usually this is because you have set the parameter but not copied the files from an HDFS node to your Vertica node. See "Additional Requirements for Kerberos" in [Configuring Hadoop and Vertica to Enable Backup of HDFS Storage](#) .

Using the HDFS Connector

Deprecated: The HDFS Connector has been deprecated and will be removed in a future release. Use the `hdfs` URL scheme instead. See [Reading Directly from HDFS](#).

The Hadoop Distributed File System (HDFS) is the location where Hadoop usually stores its input and output files. It stores files across the Hadoop cluster redundantly, to keep the files available even if some nodes are down. HDFS also makes Hadoop more efficient, by spreading file access tasks across the cluster to help limit I/O bottlenecks.

The HDFS Connector lets you load files from HDFS into Vertica using the `COPY` statement. You can also create external tables that access data stored on HDFS as if it were a native Vertica table. The connector is useful if your Hadoop job does not directly store its data in Vertica or if you want to use User-Defined Extensions (UDxs) to load data stored in HDFS.

Note: The files you load from HDFS using the HDFS Connector usually have a delimited format. Column values are separated by a character, such as a comma or a pipe character (`|`). This format is the same type used in other files you load with the `COPY` statement. Hadoop MapReduce jobs often output tab-delimited files.

The HDFS Connector takes advantage of the distributed nature of both Vertica and Hadoop. Individual nodes in the Vertica cluster connect directly to nodes in the Hadoop cluster when you load multiple files from HDFS.

Hadoop splits large files into file segments that it stores on different nodes. The connector directly retrieves these file segments from the node storing them, rather than relying on the Hadoop cluster to reassemble the file.

The connector is read-only; it cannot write data to HDFS.

The HDFS Connector can connect to a Hadoop cluster through unauthenticated and Kerberos-authenticated connections.

HDFS Connector Requirements

Uninstall Prior Versions of the HDFS Connector

The HDFS Connector is now installed with Vertica; you no longer need to download and install it separately. If you have previously downloaded and installed this connector, uninstall it

before you upgrade to this release of Vertica to get the newest version.

WebHDFS Requirements

The HDFS Connector connects to the Hadoop file system using WebHDFS, a built-in component of HDFS that provides access to HDFS files to applications outside of Hadoop. This component must be enabled on your Hadoop cluster. See your Hadoop distribution's documentation for instructions on configuring and enabling WebHDFS.

Note: HTTPfs (also known as HOOP) is another method of accessing files stored in HDFS. It relies on a separate server process that receives requests for files and retrieves them from HDFS. Since it uses a REST API that is compatible with WebHDFS, it could theoretically work with the connector. However, the connector has not been tested with HTTPfs and Micro Focus does not support using the HDFS Connector with HTTPfs. In addition, since all of the files retrieved from HDFS must pass through the HTTPfs server, it is less efficient than WebHDFS, which lets Vertica nodes directly connect to the Hadoop nodes storing the file blocks.

Kerberos Authentication Requirements

The HDFS Connector can connect to HDFS using Kerberos authentication. To use Kerberos, you must meet these additional requirements:

- Your Vertica installation must be Kerberos-enabled.
- Your Hadoop cluster must be configured to use Kerberos authentication.
- Your connector must be able to connect to the Kerberos-enabled Hadoop cluster.
- The Kerberos server must be running version 5.
- The Kerberos server must be accessible from every node in your Vertica cluster.
- You must have Kerberos principals (users) that map to Hadoop users. You use these principals to authenticate your Vertica users with the Hadoop cluster.

Testing Your Hadoop WebHDFS Configuration

To ensure that your Hadoop installation's WebHDFS system is configured and running, follow these steps:

1. Log into your Hadoop cluster and locate a small text file on the Hadoop filesystem. If you do not have a suitable file, you can create a file named `test.txt` in the `/tmp` directory using the following command:

```
echo -e "A|1|2|3\nB|4|5|6" | hadoop fs -put - /tmp/test.txt
```

2. Log into a host in your Vertica database using the database administrator account.
3. If you are using Kerberos authentication, authenticate with the Kerberos server using the keytab file for a user who is authorized to access the file. For example, to authenticate as an user named `exampleuser@MYCOMPANY.COM`, use the command:

```
$ kinit exampleuser@MYCOMPANY.COM -k -t /path/exampleuser.keytab
```

Where *path* is the path to the keytab file you copied over to the node. You do not receive any message if you authenticate successfully. You can verify that you are authenticated by using the `klist` command:

```
$ klistTicket cache: FILE:/tmp/krb5cc_500
Default principal: exampleuser@MYCOMPANY.COM
Valid starting Expires Service principal
07/24/13 14:30:19 07/25/13 14:30:19 krbtgt/MYCOMPANY.COM@MYCOMPANY.COM
renew until 07/24/13 14:30:19
```

4. Test retrieving the file:
 - If you are not using Kerberos authentication, run the following command from the Linux command line:

```
curl -i -L
"http://hadoopNameNode:50070/webhdfs/v1/tmp/test.txt?op=OPEN&user.name=hadoopUserName"
```

Replacing *hadoopNameNode* with the hostname or IP address of the name node in your Hadoop cluster, */tmp/test.txt* with the path to the file in the Hadoop filesystem you located in step 1, and *hadoopUserName* with the user name of a Hadoop user that has read access to the file.

If successful, the command produces output similar to the following:

```
HTTP/1.1 200 OKServer: Apache-Coyote/1.1
Set-Cookie:
hadoop.auth="u=hadoopUser&p=password&t=simple&e=1344383263490&s=n8YB/CHFg56qNmRQRTq00IdRMvE
="; Version=1; Path=/
Content-Type: application/octet-stream
Content-Length: 16
Date: Tue, 07 Aug 2012 13:47:44 GMT
A|1|2|3
B|4|5|6
```

- If you are using Kerberos authentication, run the following command from the Linux command line:

```
curl --negotiate -i -L -u:anyUser http://hadoopNameNode:50070/webhdfs/v1/tmp/test.txt?op=OPEN
```

Replace *hadoopNameNode* with the hostname or IP address of the name node in your Hadoop cluster, and */tmp/test.txt* with the path to the file in the Hadoop filesystem you located in step 1.

If successful, the command produces output similar to the following:

```
HTTP/1.1 401 UnauthorizedContent-Type: text/html; charset=utf-8
WWW-Authenticate: Negotiate
Content-Length: 0
Server: Jetty(6.1.26)
HTTP/1.1 307 TEMPORARY_REDIRECT
Content-Type: application/octet-stream
Expires: Thu, 01-Jan-1970 00:00:00 GMT
Set-Cookie: hadoop.auth="u=exampleuser&p=exampleuser@MYCOMPANY.COM&t=kerberos&
e=1375144834763&s=iY52iRvjuuoZ5iYG8G5g1202Vwo=";Path=/
Location:
http://hadoopnamenode.mycompany.com:1006/webhdfs/v1/user/release/docexample/test.txt?
op=OPEN&delegation=JAAHcmVsZWFzZQdyZWx1YXN1AioBQCrfpdGKAUB07CnRju3TbBS1ID_osB658jfGf
RpEt8-u9WHymRJXRUIJIREZTIGR1bGVnYXRpb24SMTAuMjAuMTAwLjlkxOjUwMDcw&offset=0
Content-Length: 0
Server: Jetty(6.1.26)
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 16
Server: Jetty(6.1.26)
A|1|2|3
B|4|5|6
```

If the curl command fails, you must review the error messages and resolve any issues before using the Vertica Connector for HDFS with your Hadoop cluster. Some debugging steps include:

- Verify the HDFS service's port number.
- Verify that the Hadoop user you specified exists and has read access to the file you are attempting to retrieve.

Loading Data Using the HDFS Connector

You can use the HDFS User Defined Source (UDS) in a [COPY](#) statement to load data from HDFS files.

The syntax for using the HDFS UDS in a COPY statement is:

```
COPY tableName SOURCE Hdfs(url='WebHDFSFileURL', [username='username'],  
[low_speed_limit=speed]);
```

<i>tableName</i>	The name of the table to receive the copied data.
<i>WebHDFSFileURL</i>	<p>A string containing one or more URLs that identify the file or files to be read. See below for details. Use commas to separate multiple URLs .</p> <p>If a URL contains certain special characters, you must escape them:</p> <ul style="list-style-type: none">• Replace any commas in the URLs with the escape sequence %2c. For example, if you are loading a file named <code>doe, john.txt</code>, change the file's name in the URL to <code>doe%2cjohn.txt</code>.• Replace any single quotes with the escape sequence ' '. For example, if you are loading a file named <code>john's_notes.txt</code>, change the file's name in the URL to <code>john' 's_notes.txt</code>.
<i>username</i>	The username of a Hadoop user that has permissions to access the files you want to copy. If you are using Kerberos, omit this argument.
<i>speed</i>	The minimum data transmission rate, expressed in bytes per second, that the connector allows. The connector breaks any connection between the Hadoop and Vertica clusters that transmits data slower than this rate for more than 1 minute. After the connector breaks a connection for being too slow, it attempts to connect to another node in the Hadoop cluster. This new connection can supply the data that the broken connection was retrieving. The connector terminates the COPY statement and returns an error message if:

	<ul style="list-style-type: none"> • It cannot find another Hadoop node to supply the data. • The previous transfer attempts from all other Hadoop nodes that have the file also closed because they were too slow. <p>Default Value: 1048576 (1MB per second transmission rate)</p>
--	---

The HDFS File URL

The *url* parameter in the Hdfs function call is a string containing one or more comma-separated HTTP URLs. These URLs identify the files in HDFS that you want to load. The format for each URL in this string is:

```
http://NameNode:port/webhdfs/v1/HDFSFilePath
```

<i>NameNode</i>	The host name or IP address of the Hadoop cluster's name node.
<i>Port</i>	The port number on which the WebHDFS service is running. This number is usually 50070 or 14000, but may be different in your Hadoop installation.
<i>webhdfs/v1/</i>	The protocol being used to retrieve the file. This portion of the URL is always the same. It tells Hadoop to use version 1 of the WebHDFS API.
<i>HDFSFilePath</i>	The path from the root of the HDFS filesystem to the file or files you want to load. This path can contain standard Linux wildcards. <div style="border: 1px solid gray; padding: 5px; margin-top: 10px;"> <p>Important: Any wildcards you use to specify multiple input files must resolve to files only. They must not include any directories. For example, if you specify the path <code>/user/HadoopUser/output/*</code>, and the output directory contains a subdirectory, the connector returns an error message.</p> </div>

The following example shows how to use the Vertica Connector for HDFS to load a single file named `/tmp/test.txt`. The Hadoop cluster's name node is named `hadoop`.

```
=> COPY testTable SOURCE Hdfs(url='http://hadoop:50070/webhdfs/v1/tmp/test.txt',
  username='hadoopUser');
  Rows Loaded
  -----
                2
(1 row)
```

Copying Files in Parallel

The basic COPY statement in the previous example copies a single file. It runs on just a single host in the Vertica cluster because the Connector cannot break up the workload among nodes. Any data load that does not take advantage of all nodes in the Vertica cluster is inefficient.

To make loading data from HDFS more efficient, spread the data across multiple files on HDFS. This approach is often natural for data you want to load from HDFS. Hadoop MapReduce jobs usually store their output in multiple files.

You specify multiple files to be loaded in your Hdfs function call by:

- Using wildcards in the URL
- Supplying multiple comma-separated URLs in the url parameter of the Hdfs user-defined source function call
- Supplying multiple comma-separated URLs that contain wildcards

Loading multiple files through the Vertica Connector for HDFS results in a efficient load. The Vertica hosts connect directly to individual nodes in the Hadoop cluster to retrieve files. If Hadoop has broken files into multiple chunks, the Vertica hosts directly connect to the nodes storing each chunk.

The following example shows how to load all of the files whose filenames start with "part-" located in the /user/hadoopUser/output directory on the HDFS. If there are at least as many files in this directory as there are nodes in the Vertica cluster, all nodes in the cluster load data from the HDFS.

```
=> COPY Customers SOURCE-> Hdfs(url='http://hadoop:50070/webhdfs/v1/user/hadoopUser/output/part-*',
  username='hadoopUser');
  Rows Loaded
  -----
          40008
(1 row)
```

To load data from multiple directories on HDFS at once use multiple comma-separated URLs in the URL string:

```
=> COPY Customers SOURCE-> Hdfs(url='http://hadoop:50070/webhdfs/v1/user/HadoopUser/output/part-*,
  http://hadoop:50070/webhdfs/v1/user/AnotherUser/part-*',
  username='H=hadoopUser');
  Rows Loaded
  -----
          80016
(1 row)
```

Note: Vertica statements must be less than 65,000 characters long. If you supply too many long URLs in a single statement, you could go over this limit. Normally, you would only approach this limit if you are automatically generating of the COPY statement using a program or script.

Viewing Rejected Rows and Exceptions

COPY statements that use the Vertica Connector for HDFS use the same method for recording rejections and exceptions as other COPY statements. Rejected rows and exceptions are saved to log files. These log files are stored by default in the CopyErrorLogs subdirectory in the database's catalog directory. Due to the distributed nature of the Vertica Connector for HDFS, you cannot use the ON option to force all exception and rejected row information to be written to log files on a single Vertica host. Instead, you need to collect the log files from across the hosts to review all of the exceptions and rejections generated by the COPY statement.

For more about handling rejected rows, see [Capturing Load Rejections and Exceptions](#).

Creating an External Table with an HDFS Source

You can use the HDFS Connector as a source for an external table that lets you directly perform queries on the contents of files on the Hadoop Distributed File System (HDFS). See [Using External Tables](#) in the Administrator's Guide for more information on external tables. If your HDFS data is in ORC or Parquet format, using the special readers for those formats might provide better performance. See [Reading Hadoop Columnar File Formats](#).

Using an external table to access data stored on an HDFS cluster is useful when you need to extract data from files that are periodically updated, or have additional files added on HDFS. It saves you from having to drop previously loaded data and then reload the data using a COPY statement. The external table always accesses the current version of the files on HDFS.

Note: An external table performs a bulk load each time it is queried. Its performance is significantly slower than querying an internal Vertica table. You should only use external tables for infrequently-run queries (such as daily reports). If you need to frequently query the content of the HDFS files, you should either use COPY to load the entire content of the files into Vertica or save the results of a query run on an external table to an internal table which you then use for repeated queries.

To create an external table that reads data from HDFS, use the HDFS Use-Defined Source (UDS) in a [CREATE EXTERNAL TABLE AS COPY](#) statement. The COPY portion of this statement has the same format as the COPY statement used to load data from HDFS. See [Loading Data Using the HDFS Connector](#) for more information.

The following simple example shows how to create an external table that extracts data from every file in the `/user/hadoopUser/example/output` directory using the HDFS Connector.

```
=> CREATE EXTERNAL TABLE hadoopExample (A VARCHAR(10), B INTEGER, C INTEGER, D INTEGER)
-> AS COPY SOURCE Hdfs(url=
-> 'http://hadoop01:50070/webhdfs/v1/user/hadoopUser/example/output/*',
-> username='hadoopUser');
CREATE TABLE
=> SELECT * FROM hadoopExample;
  A  | B | C | D
-----+-----+-----+-----
test1 | 1 | 2 | 3
test1 | 3 | 4 | 5
(2 rows)
```

Later, after another Hadoop job adds contents to the output directory, querying the table produces different results:

```
=> SELECT * FROM hadoopExample;
  A  | B | C | D
-----+-----+-----+-----
test3 | 10 | 11 | 12
test3 | 13 | 14 | 15
test2 | 6 | 7 | 8
test2 | 9 | 0 | 10
test1 | 1 | 2 | 3
test1 | 3 | 4 | 5
(6 rows)
```

Load Errors in External Tables

Normally, querying an external table on HDFS does not produce any errors if rows rejected by the underlying COPY statement (for example, rows containing columns whose contents are incompatible with the data types in the table). Rejected rows are handled the same way they are in a standard COPY statement: they are written to a rejected data file, and are noted in the exceptions file. For more information on how COPY handles rejected rows and exceptions, see [Capturing Load Rejections and Exceptions](#) in the Administrator's Guide.

Rejections and exception files are created on all of the nodes that load data from the HDFS. You cannot specify a single node to receive all of the rejected row and exception information. These files are created on each Vertica node as they process files loaded through the Vertica Connector for HDFS.

Note: Since the the connector is read-only, there is no way to store rejection and exception information on the HDFS.

Fatal errors during the transfer of data (for example, specifying files that do not exist on the HDFS) do not occur until you query the external table. The following example shows what happens if you recreate the table based on a file that does not exist on HDFS.

```
=> DROP TABLE hadoopExample;
DROP TABLE
=> CREATE EXTERNAL TABLE hadoopExample (A INTEGER, B INTEGER, C INTEGER, D INTEGER)
-> AS COPY SOURCE HDFS(url='http://hadoop01:50070/webhdfs/v1/tmp/nofile.txt',
-> username='hadoopUser');
CREATE TABLE
=> SELECT * FROM hadoopExample;
ERROR 0: Error calling plan() in User Function HdfsFactory at
[src/Hdfs.cpp:222], error code: 0, message: No files match
[http://hadoop01:50070/webhdfs/v1/tmp/nofile.txt]
```

Note that it is not until you actually query the table that the connector attempts to read the file. Only then does it return an error.

HDFS Connector Troubleshooting Tips

The following sections explain some of the common issues you may encounter when using the HDFS Connector.

User Unable to Connect to Kerberos-Authenticated Hadoop Cluster

A user may suddenly be unable to connect to Hadoop through the connector in a Kerberos-enabled environment. This issue can be caused by someone exporting a new keytab file for the user, which invalidates existing keytab files. You can determine if invalid keytab files is the problem by comparing the key version number associated with the user's principal key in Kerberos with the key version number stored in the keytab file on the Vertica cluster.

To find the key version number for a user in Kerberos:

1. From the Linux command line, start the kadmin utility (kadmin.local if you are logged into the Kerberos Key Distribution Center). Run the getprinc command for the user:

```
$ sudo kadmin
[sudo] password for dbadmin:
Authenticating as principal root/admin@MYCOMPANY.COM with password.
Password for root/admin@MYCOMPANY.COM:
kadmin: getprinc exampleuser@MYCOMPANY.COM
Principal: exampleuser@MYCOMPANY.COM
Expiration date: [never]
Last password change: Fri Jul 26 09:40:44 EDT 2013
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 0 days 00:00:00
Last modified: Fri Jul 26 09:40:44 EDT 2013 (root/admin@MYCOMPANY.COM)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 2
Key: vno 3, des3-cbc-sha1, no salt
Key: vno 3, des-cbc-crc, no salt
MKey: vno 0
Attributes:
Policy: [none]
```

In the preceding example, there are two keys stored for the user, both of which are at version number (vno) 3.

2. To get the version numbers of the keys stored in the keytab file, use the klist command:

```
$ sudo klist -ek exampleuser.keytab
Keytab name: FILE:exampleuser.keytab
KVNO Principal
-----
 2 exampleuser@MYCOMPANY.COM (des3-cbc-sha1)
 2 exampleuser@MYCOMPANY.COM (des-cbc-crc)
 3 exampleuser@MYCOMPANY.COM (des3-cbc-sha1)
 3 exampleuser@MYCOMPANY.COM (des-cbc-crc)
```

The first column in the output lists the key version number. In the preceding example, the keytab includes both key versions 2 and 3, so the keytab file can be used to authenticate the user with Kerberos.

Resolving Error 5118

When using the connector, you might receive an error message similar to the following:

```
ERROR 5118: UDL specified no execution nodes; at least one execution node must be specified
```

To correct this error, verify that all of the nodes in your Vertica cluster have the correct version of the HDFS Connector package installed. This error can occur if one or more of the nodes do not have the supporting libraries installed. These libraries may be missing because one of the

nodes was skipped when initially installing the connector package. Another possibility is that one or more nodes have been added since the connector was installed.

Transfer Rate Errors

The HDFS Connector monitors how quickly Hadoop sends data to Vertica. In some cases, the data transfer speed on any connection between a node in your Hadoop cluster and a node in your Vertica cluster slows beyond a lower limit (by default, 1 MB per second). When the transfer slows beyond the lower limit, the connector breaks the data transfer. It then connects to another node in the Hadoop cluster that contains the data it was retrieving. If it cannot find another node in the Hadoop cluster to supply the data (or has already tried all of the nodes in the Hadoop cluster), the connector terminates the COPY statement and returns an error.

```
=> COPY messages SOURCE Hdfs(url='http://hadoop.example.com:50070/webhdfs/v1/tmp/data.txt',
    username='exampleuser');
ERROR 3399: Failure in UDx RPC call InvokeProcessUDL(): Error calling processUDL()
in User Defined Object [Hdfs] at [src/Hdfs.cpp:275], error code: 0,
message: [Transferring rate during last 60 seconds is 172655 byte/s, below threshold 1048576 byte/s,
give up.
The last error message: Operation too slow. Less than 1048576 bytes/sec transferred the last 1
seconds.
The URL:
http://hadoop.example.com:50070/webhdfs/v1/tmp/data.txt?op=OPEN&offset=154901544&length=113533912.
The redirected URL: http://hadoop.example.com:50075/webhdfs/v1/tmp/data.txt?op=OPEN&
namenoderpcaddress=hadoop.example.com:8020&length=113533912&offset=154901544.]
```

If you encounter this error, troubleshoot the connection between your Vertica and Hadoop clusters. If there are no problems with the network, determine if either your Hadoop cluster or Vertica cluster is overloaded. If the nodes in either cluster are too busy, they may not be able to maintain the minimum data transfer rate.

If you cannot resolve the issue causing the slow transfer rate, you can lower the minimum acceptable speed. To do so, set the `low_speed_limit` parameter for the Hdfs source. The following example shows how to set `low_speed_limit` to 524288 to accept transfer rates as low as 512 KB per second (half the default lower limit).

```
=> COPY messages SOURCE Hdfs(url='http://hadoop.example.com:50070/webhdfs/v1/tmp/data.txt',
    username='exampleuser', low_speed_limit=524288);
Rows Loaded
-----
      9891287
(1 row)
```

When you lower the `low_speed_limit` parameter, the COPY statement loading data from HDFS may take a long time to complete.

You can also increase the `low_speed_limit` setting if the network between your Hadoop cluster and Vertica cluster is fast. You can choose to increase the lower limit to force COPY statements

to generate an error, if they are running more slowly than they normally should, given the speed of the network.

Error Loading Many Files

When using the HDFS Connector to load many data files in a single statement, you might receive an error message similar to the following:

```
RROR 3399: Failure in UDX RPC call InvokePlanUDL():  
Error calling planUDL() in User Defined Object [Hdfs] at [src/Glob.cpp:531],  
error code: 0, message:  
Error occurs in Glob::stat:  
Last error message before give up: Failed to connect to 10.20.41.212:  
Cannot assign requested address.
```

This can happen when concurrent load requests overwhelm the Name Node. It is generally safe to load hundreds of files at a time, but if you load thousands you might see this error. Use smaller batches of files to avoid this error.

Integrating With Cloudera Manager

The Cloudera distribution of Hadoop includes Cloudera Manager, a web-based tool for managing a Hadoop cluster. Cloudera Manager can manage any service for which a service description is available, including Vertica.

You can use Cloudera Manager to start, stop, and monitor individual database nodes or the entire database. You can manage both co-located and separate Vertica clusters—Cloudera can manage services on nodes that are not part of the Hadoop cluster.

You must install and configure your Vertica database before proceeding; you cannot use Cloudera Manager to create the database.

Installing the Service

Note: Because the service has to send the database password over the network, you should enable encryption on your Hadoop cluster before proceeding.

A Cloudera Service Description (CSD) file describes a service that Cloudera can manage. The Vertica CSD is in `/opt/vertica/share/CSD` on a database node.

To install the Vertica CSD, follow these steps:

1. On a Vertica node, follow the instructions in [VerticaAPIKey](#) to generate an API key. You need this key to finish the installation of the CSD.
2. On the Hadoop node that hosts Cloudera Manager, copy the CSD file into `/opt/cloudera/csd`.
3. Restart Cloudera Manager:

```
$ service cloudera-scm-server restart
```

4. In a web browser, go to Cloudera Manager and restart the Cloudera Management Service.
5. If your Vertica cluster is separate from your Hadoop cluster (not co-located on it): Use Cloudera Manager to add the hosts for your database nodes. If your cluster is co-located, skip this step.
6. Use Cloudera Manager to add the Vertica service.

7. On the "Role Assignment" page, select the hosts that are database nodes.
8. On the "Configuration" page, specify values for the following fields:
 - database name
 - agent port (accept the default if you're not sure)
 - API key
 - database user to run as (usually dbadmin) and password

About the Agent

When you manage Vertica through Cloudera Manager, you are actually interacting with the Vertica Agent, not the database directly. The [Agent](#) runs on all database nodes and interacts with the database on your behalf. Management Console uses the same agent. Most of the time this extra indirection is transparent to you.

A Cloudera-managed service contains one or more roles. In this case the service is "Vertica" and the single role is "Vertica Node".

Available Operations

Cloudera Manager shows two groups of operations. Service-level operations apply to the service on all nodes, while role-level operations apply only to a single node.

You can perform the following service-level operations on all nodes:

- Start: Starts the agent and, if it is not already running, the database.
- Stop: Stops the database and agent.
- Restart: Calls Stop and then Start.
- Add Role Instances: Adds new database nodes to Cloudera Manager. The nodes must already be part of the Vertica cluster, and the hosts must already be known to Cloudera Manager.
- Enter Maintenance Mode: Suppresses health alerts generated by Cloudera Manager.
- Exit Maintenance Mode: Resumes normal reporting.

- **Update Memory Pool Size:** Applies memory-pool settings from the Static Service Pools configuration page.

You can perform all of these operations except Add Role Instances on individual nodes as role-level operations.

Managing Memory Pools

Cloudera Manager allows you to change resource allocations, such as memory and CPU, for the nodes it manages. If you are using co-located clusters, centrally managing resources can simplify your cluster management. If you are using separate Hadoop and Vertica clusters, you might prefer to manage Vertica separately as described in [Managing the Database](#) in the Administrator's Guide.

Use the Cloudera Manager "Static Service Pools" configuration page to configure resource allocations. The "Vertica Memory Pool" value, specified in GB, is the maximum amount of memory to allocate to the database on each node. If the configuration page includes "Cgroup Memory Hard Limit", set it to the same value as "Vertica Memory Pool".

After you have set these values, you can use the "Update Memory Pool Size" operation to apply the value to the managed nodes. This operation is equivalent to `ALTER RESOURCE POOL GENERAL MAXMEMORYSIZE`. Configuration changes in "Static Service Pools" do not take effect in Vertica until you perform this operation.

Uninstalling the Service

To uninstall the Vertica CSD, follow these steps:

1. Stop the Vertica service and then remove it from Cloudera Manager.
2. Remove the CSD file from `/opt/cloudera/csd`.
3. From the command line, restart the Cloudera Manager server.
4. In Cloudera Manager, restart the Cloudera Management Service.

Integrating Vertica with the MapR Distribution of Hadoop

MapR is a distribution of Apache Hadoop produced by MapR Technologies that extends the standard Hadoop components with its own features. Vertica can integrate with MapR in the following ways:

- You can read data from MapR through an NFS mount point. After you mount the MapR file system as an NFS mount point, you can use [CREATE EXTERNAL TABLE AS COPY](#) or [COPY](#) to access the data as if it were on the local file system. For ORC or Parquet data, see also [Reading Hadoop Columnar File Formats](#). This option provides the best performance for reading data.
- You can use the HCatalog Connector to read Hive data. Do not use the HCatalog Connector with ORC or Parquet data in MapR for performance reasons. Instead, mount the MapR file system as an NFS mount point and create external tables without using the Hive schema. See [Using the HCatalog Connector](#).
- You can create a storage location to store data in MapR using the native Vertica format (ROS). Mount the MapR file system as an NFS mount point and then use [CREATE LOCATION...ALL NODES SHARED](#) to create a storage location. (CREATE LOCATION does not support NFS mount points in general, but does support them for MapR.)

Other Vertica integrations for Hadoop are not available for MapR.

For information on mounting the MapR file system as an NFS mount point, see [Accessing Data with NFS](#) and [Configuring Vertica Analytics Platform with MapR](#) on the MapR website. In particular, you must configure MapR to add Vertica as a MapR service.

Examples

In the following examples, the MapR file system has been mounted as `/mapr`.

The following statement creates an external table from ORC data:

```
=> CREATE EXTERNAL TABLE t (a1 INT, a2 VARCHAR(20))  
AS COPY FROM '/mapr/data/file.orc' ORC;
```

The following statement creates an external table from Parquet data and takes advantage of partition pruning (see [Using Partition Columns](#)):

```
=> CREATE EXTERNAL TABLE t2 (id int, name varchar(50), created date, region varchar(50))  
    AS COPY FROM '/mapr/**/*' PARQUET(hive_partition_cols='created,region');
```

The following statement loads ORC data from MapR into Vertica:

```
=> COPY t FROM '/mapr/data/*.orc' ON ANY NODE ORC;
```

The following statement creates a storage location to hold ROS data in the MapR file system:

```
=> CREATE LOCATION '/mapr/my.cluster.com/data' SHARED USAGE 'DATA' LABEL 'maprfs';
```


Integrating with Apache Kafka

Welcome to the Vertica Data Streaming Integration Guide.

Audience

This book is intended for anyone who wants to load data from an existing data streaming message bus into a Vertica database.

Prerequisites

This document assumes that you have installed and configured Vertica as described in [Installing Vertica](#) and the Configuring the Database section of the Administrator's Guide. In addition, you must have Java 7.0 or later installed on your Vertica node. Refer to the [Vertica product documentation](#) to learn more.

In addition, this guide assumes you have installed and configured your data streaming platform. For details on installing and using third party applications, please refer to the documentation for that application.

How Vertica and Data Streaming Work Together

Vertica provides a high-performance loading mechanism for streaming data from a third party message bus into your Vertica database.

Data streaming provides high volumes of data with low latency. Vertica can process this data by running many COPY statements, each of which loads small amounts of data into your Vertica database. However, this process can become complex. Instead of writing complicated ETL processes and dispatching COPY statements manually, you can use the data streaming integration feature to automatically load data as it streams through your message bus.

The integration features between Vertica and data streaming consist of:

- A UDL library that loads data from a message bus into Vertica
- A job scheduler that uses the UDL library to continuously consume data from your message bus with exactly-once semantics
- Push-based [Monitoring Vertica Using Notifiers](#) capable of sending messages from Vertica to Kafka
- A [KafkaExport](#) function that outputs rows that Vertica was unable to send to Kafka

This process contributes to low latency and minimal impact on the other processes on the database.

Data Streaming Integration Terms

Vertica integrates with data streaming applications through a number of components. To use Vertica with data streaming, you should be familiar with these terms.

Terminology

Term	Description
Host	A data streaming server.
Source	A feed of messages in a common category which streams into the same Vertica target tables. In Apache Kafka, a source is known as a topic.
Partition	Unit of parallelism within data streaming. Data streaming splits a source into multiple partitions, which can each be served in parallel to consumers such as a Vertica database. Within a partition, all messages are ordered chronologically.
Offset	An index into a partition. This index is the position within an ordered queue of messages, not an index into an opaque byte stream.
Message	A unit of data within data streaming. The data is typically in JSON or AVRO format. Messages are loaded as rows into Vertica tables, and are uniquely identified by their source, partition, and offset.

Data Loader Terminology

Data Loader Term	Description
Job scheduler	A tool for continuous loading of data from data streaming into Vertica.
Micro-batch	A pair of statements that: a) load data from all sources

Data Loader Term	Description
	configured for this micro-batch into a Vertica target table; and b) update the progress within the streams. Being an atomic transaction, the micro-batch rolls back if any part of these operations fail so that each message is loaded exactly once.
Frame	Duration of time in which the scheduler will attempt to load each configured source once.
Stream	A feed of messages that is identified by a source and partition. The offset uniquely identifies the position within a particular source-partition stream.
Lane	A thread within a job scheduler instance that issues micro-batches to perform the load. The number of lanes available is based on the <code>PlannedConcurrency</code> of the job scheduler's resource pool. Multiple lanes allow for parallelism of micro-batches during a frame.

Data Streaming Job Scheduler

The data streaming *job scheduler* is a tool for continuous loading of streaming data into Vertica. The scheduler comes pre-packaged and installed with the Vertica rpm. For information on job scheduler requirements, refer to [Vertica Integration for Apache Kafka](#).

You can use the scheduler from any node by running the vkconfig script:

```
/opt/vertica/packages/kafka/bin/vkconfig
```

Note: If you do not want the scheduler to use Vertica host resources, or if you want to limit user access to the Vertica nodes, install the RPM on the host but do not create a database.

What the Scheduler Does

A scheduler instance works by creating frames and issuing micro-batches that load data into tables in your Vertica database. The scheduler loads all (enabled) sources to Vertica target tables during a single frame duration and continuously schedules frames as one completes.

You can add as many sources as you want to a single scheduler. Doing so allows the scheduler to collect all data from all these sources every single frame. This option is helpful if you have a large number of sources.

What Happens When You Create a Scheduler

When you create a new scheduler, the following events occur:

- The script creates a new Vertica schema with a name you specify (default is `stream_config`). You use this name to identify the scheduler during configuration.
- The script creates [Data Streaming Schema Tables](#) for the Vertica schema.
- The script creates the resource pool `kafka_default_pool`, if it does not already exist.

When the script creates the schema and associated tables, it sets the `LOCKTIMEOUT` session configuration parameter to 0 for the session running the micro-batches. When `LOCKTIMEOUT` is 0, data loads continuously because the scheduler does not have to wait for a lock to be released. If a table is already locked, Vertica cancels the frame and records an error in the events table.

The script creates the resource pool with defaults that benefit loading data into Vertica. While you can alter this pool to your business needs, Micro Focus strongly recommends following these guidelines:

- Leave the `QUEUE_TIMEOUT` parameter set to 0. This value is the default for job scheduler resource pools. A value of 0 allows data to load continuously. If the scheduler has to wait for resources, it cannot progress, compromising scheduling configurations.
- Leave reflexive moveout enabled. This option is on automatically when you create a schedule. With reflexive moveout turned on, the Tuple Mover automatically performs a moveout operation when data is committed so that your WOS always has space to load data. For large volumes of data (>100 MB) use a load method of `DIRECT`.

Validating Schedulers

When you create or configure a scheduler, Vertica validates the settings that you provide. Vertica checks the following settings:

- Confirms that all brokers in the specified cluster exist.
- Confirms that the specified source exists.
- Compares the host being configured to all existing cluster hosts. If the host already exists, Vertica cancels the configuration.
- Verifies that the number of partitions equals the number provided by the user. If no number of partitions is specified, Vertica sets the value to the number of partitions the source has in the cluster.
- Compares the cluster host list to the cluster being configured. If the cluster already exists, Vertica cancels the configuration.

You can configure validation checking using the `--validation-type` parameter in [Scheduler Utility Options](#).

Synchronizing Schedulers

By default, Vertica automatically synchronizes source information with host clusters. You can configure the synchronization interval using the `--refresh-config` Scheduler Utility Option. Vertica synchronizes the following settings:

- Updates the broker list for each cluster.
- Confirms that each source exists. If a source does not exist, Vertica disables it. If Vertica cannot reach the cluster at all, it takes no action.
- Updates the number of partitions for each source.

You can configure synchronization settings using the `--auto-sync` parameter in [Scheduler Utility Options](#).

Launching a Scheduler

To launch a scheduler, you must have a running streaming instance in a place Vertica can access. Additionally, you must configure the scheduler and set up sources for streaming.

When you launch a scheduler, the scheduler collects data from your sources, starting at the specified offset. You can view the [stream_microbatch_history](#) table to see what the scheduler is doing at any given time.

To learn how to create, configure, and launch a scheduler, see [Using Streaming Data with Vertica](#) in this guide.

You can also choose to bypass the scheduler. For example, you might want to do a single load with a specific range of offsets. For more information, see [Using COPY with Data Streaming](#) in this guide.

Launching Multiple Schedulers for High Availability

For high availability, you can launch two or more identical schedulers that target the same configuration schema. You can differentiate these different schedulers using the `--instance-name` CLI option with the [Launch Utility Options](#). The scheduler not in use remains in stand-by mode and can only perform scheduling if the active scheduler fails. In this case, the stand-by process allows the stream to continue without interruption.

Viewing Schedulers from the MC

You can also view the status of Kafka jobs from the MC. For more information, refer to [Viewing Load History](#).

Updating Schedulers After Vertica Upgrades

A scheduler is only compatible with the version of Vertica that created it. When you upgrade Vertica to a new major version or service pack, you must also update your schedulers using the `--upgrade` option before you can restart them. If you do not update a scheduler, you receive an error message if you try to launch it. For example:

```
$ vkconfig launch --conf weblog.conf
com.vertica.solutions.kafka.exception.FatalException: Configured scheduler schema and current
scheduler configuration schema version do not match. Upgrade configuration by running: vkconfig
scheduler --upgrade
    at com.vertica.solutions.kafka.scheduler.StreamCoordinator.assertVersion(StreamCoordinator.java:54)
    at com.vertica.solutions.kafka.scheduler.StreamCoordinator.run(StreamCoordinator.java:125)
    at com.vertica.solutions.kafka.Launcher.run(Launcher.java:205)
    at com.vertica.solutions.kafka.Launcher.main(Launcher.java:258)
Scheduler instance failed. Check log file. Check log file.
$ vkconfig scheduler --upgrade --conf weblog.conf
Checking if UPGRADE necessary...
UPGRADE required, running UPGRADE...
UPGRADE completed successfully, now the scheduler configuration schema version is v8.1.1
$ vkconfig launch --conf weblog.conf
. . .
```

Using Streaming Data with Vertica

To begin using streaming data and Vertica, use the `vkconfig` script to complete the following tasks from your Vertica database:

1. [Create a Cluster](#)
2. [Create and Configure a Scheduler](#)
3. [Create a Data Table](#)
4. [Create a Source](#)
5. [Create a Target](#)
6. [Create a Load-Spec](#)
7. [Create a Microbatch](#)
8. [Launch the Scheduler](#)

Note: You can view help for any of these steps using the command `/opt/vertica/packages/kafka/bin/vkconfig source --help`

Create a Cluster

You must associate at least one Kafka cluster with your scheduler. Vertica supports connecting multiple clusters to the same Vertica database. When you create a cluster, you identify the name of that cluster and the hosts it contains.

```
/opt/vertica/packages/kafka/bin/vkconfig cluster --create --config-  
schema kafka_conversion --cluster kafka --hosts  
1kafka01:9092,1kafka02:9092,1kafka03:9092
```

See [Cluster Utility Options](#) for more information.

Create and Configure Scheduler

Vertica includes a default `stream_config` scheduler. You can use this scheduler or create a new scheduler using the `vkconfig` script with the `scheduler` utility and `--create` option:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --create
```

The `--create` option is all that is required to add a scheduler with default options.

You can use additional configuration parameters to further customize your scheduler.

The following example shows how you can use the commands to:

- Create a scheduler called "myScheduler" with the `--config-schema` option.
- Grant privileges to run the scheduler to `kafka_user` with the `--operator` option. The `dbadmin` user must specify kafka's additional privileges separately.
- Specify a frame duration of thirty seconds with the `--frame-duration` option.

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --create --  
config-schema myScheduler --operator kafka_user --frame-duration  
'00:00:30'
```

See [Scheduler Utility Options](#) for more information.

Create a Data Table

Before configuring a source for streaming, create a target table in your Vertica database to store the data you capture. To load data into a flexible table, you must be [using a flex parser](#). If you are not using flexible tables, you must verify that the data you are streaming matches the columns in your target table.

```
CREATE FLEX TABLE public.kafka_tgt();
```

You do not need to create a rejection table, which stores rejected messages, because the table is created automatically when you run the Source utility.

```
/opt/vertica/packages/kafka/bin/vkconfig cluster --create --cluster  
StreamCluster1 --hosts 10.10.10.10:9092,10.10.10.11:9092
```

Create a Source

To create and associate a source with a configured scheduler, use the source sub-utility.

The following example shows how you can create a source and associate it and three of its partitions to the default "stream_config" scheduler.

```
//opt/vertica/packages/kafka/bin/vkconfig source --create --config-schema stream_config --cluster StreamCluster1 --source conversion -partitions 3
```

See [Source Utility Options](#) for more information.

Create a Target

Once you have a source, configure a target table to receive data from that source. The following example identifies a target table of openx.conversion.

```
/opt/vertica/packages/kafka/bin/vkconfig target --create --config-schema kafka_conversion --target-schema openx --target-table conversion
```

See [Target Utility Options](#) for more information.

Create a Load-Spec

A load-spec provides parameters that Vertica uses when loading streaming data. It also identifies any filters that you want to apply to your data.

```
/opt/vertica/packages/kafka/bin/vkconfig load-spec --create --config-schema kafka_conversion --load-spec kafka
```

See [Load Spec Utility Options](#) for more information.

Create a Microbatch

A microbatch is a COPY that streams data to your target. The microbatch uses the parameters provided by your load-spec. The following example creates a new microbatch and assigns the source cluster kafka to it.

```
/opt/vertica/packages/kafka/bin/vkconfig microbatch --create --  
config-schema kafka_conversion --microbatch conversion --target-  
schema openx --target-table conversion --rejection-schema openx --  
rejection-table conversion_rej --load-spec kafka --add-source  
conversion --add-source-cluster kafka
```

See [Microbatch Utility Options](#) for more information.

Launch the Scheduler

After you create a table and associate a source, you are ready to launch the scheduler and start streaming data. Launch a configured scheduler by using the launch sub-utility.

The following example launches the default scheduler, stream_config, and specifies a properties file, configFile.properties, which contains additional CLI options. To start a different scheduler, use the --config-schema parameter option.

```
/opt/vertica/packages/kafka/bin/vkconfig launch --conf  
configFile.properties
```

Important: Micro Focus International plc does not recommend specifying a password on the command line. Instead, put the password in a properties file.

See [Launch Utility Options](#) for more information.

Using COPY with Data Streaming

You can manually stream data from Kafka into Vertica using a COPY statement. This streaming happens only for a limited time; when the COPY statement finishes, no additional messages are streamed from Kafka to Vertica. Manually copying data this way is useful when you have a specific set of messages in Kafka that you want to analyze. It is also useful when you want to explore the data in a Kafka stream before setting up a scheduler to continuously stream the data into Vertica.

The source of your COPY statement is always KafkaSource. Your COPY statement usually uses one of three parsers: KafkaParser, KafkaJSONParser, or KafkaAvroParser.

For example:

```
=> COPY public.from_kafka
  SOURCE KafkaSource(stream='iot_data|0|-2,iot_data|1|-2',
                    brokers='kafka01.example.com:9092',
                    duration=interval '10000 milliseconds',
                    executionparallelism='1',
                    stop_on_eof=true)
  PARSER KafkaAVROParser(schema_registry_url='http://localhost:8081/subjects/iot_data-
value/versions/1')
  REJECTED DATA AS TABLE public.rejections
  DIRECT NO COMMIT;
```

In the previous example:

- The KafkaSource streams data from partitions 0 and 1 of the topic named `iot_data`.
- The streaming starts from the earliest available message in the stream (set by the `-2` in the stream parameter).
- KafkaSource reads the data from the Kafka broker running on the host named `kafka01.example.com` on port 9092.
- The streaming continues until either 10000 milliseconds (10 seconds) pass or KafkaSource reaches the end of the stream.
- KafkaSource is limited to using a single thread to load data from the two partitions.
- The stream is parsed as Avro data.
- The schema that the Avro parser uses to parse the data is retrieved from a schema registry running on the local system.
- Rejected data is saved in a table named `public.rejections`.

When you use COPY TRICKLE, Vertica recommends enabling the ReflexiveMoveout configuration parameter to trigger a Tuple Mover moveout task every time a commit occurs:

```
=> ALTER DATABASE dbname SET ReflexiveMoveout=1;
```

Note: If you are copying data containing default values into a flex table, you must identify the default value column as `__raw__`. For more information, refer to "Handling Default Values During Loading" in [Bulk Loading Data into Flex Tables](#).

The following sections explain the parameters you can pass to the KafkaSource and its associated parsers.

KafkaSource

The KafkaSource UDL accesses data from a Kafka cluster. All Kafka parsers must use KafkaSource. Messages processed by KafkaSource must be at least one byte in length. KafkaSource writes an error message to vertica.log for zero length messages.

The syntax for calling KafkaSource is:

```
KafkaSource(  
  stream='topic_name|partition|start_offset[|end_offset][,...]'  
  [, brokers='host:port[,...]']  
  [, duration=interval]  
  [, executionparallelism='value']  
  [, stop_on_eof=Boolean]  
  [, eof_timeout=timeout])
```

Parameter	Description
stream	<p>Required. Defines the data to be loaded as a comma-separated list of one or more partitions. Each partition is defined by three required values and one optional value separated by pipe characters ():</p> <ul style="list-style-type: none">• <i>topic_name</i>: the name of the Kafka topic to load data from. You can read from different Kafka topics in the same stream parameter, with some limitations. See Loading from Multiple Topics in the Same Stream Parameter below for more information.• <i>partition</i>: the partition in the Kafka topic to copy.

Parameter	Description
	<ul style="list-style-type: none"> • <i>start_offset</i>: the offset in the Kafka topic where the load will begin. This offset is inclusive (the message with the offset <i>start_offset</i> is loaded). • <i>end_offset</i>: the optional offset where the load should end. This offset is exclusive (the message with the offset <i>end_offset</i> will not be loaded). To end a load using <i>end_offset</i>, you must supply an ending offset value for all partitions in the stream parameter. Attempting to set an ending offset for some partitions and not and not set offset values for others results in an error. If you do not specify an ending offset, you must supply at least one other ending condition using <i>stop_on_eof</i> or <i>duration</i>.
brokers	<p>A comma-separated list of host:port pairs of the brokers in the Kafka cluster. Vertica recommends running Kafka on a different machine than Vertica.</p> <p>Default Value: localhost:9092</p>
duration	<p>An INTERVAL that specifies the duration of the frame. After this specified amount of time, KafkaSource terminates the COPY statements. If this parameter is not set, you must set at least one other ending condition by using <i>stop_on_eof</i> or specify an ending offset instead. See Duration Note below for more information.</p>
executionparallelism	<p>A integer value between 1 and the number of partitions the node is loading from. Setting this parameter to a reduced value limits the number of threads used to process any COPY statement. It also increases the throughput of short queries issued in the pool, especially if the queries are executed concurrently.</p> <p>If you do not specify this parameter, Vertica automatically creates a thread for every partition.</p> <p>If the value you specify for the KafkaSource is lower than the value specified for the scheduler resource pool, the KafkaSource value applies. This value cannot exceed the</p>

Parameter	Description
	value specified for the scheduler resource pool.
<code>stop_on_eof</code>	Determines whether <code>KafkaSource</code> should terminate the <code>COPY</code> statement after it reaches the end of a file. If this value is not set, you must set at least one other ending condition using <code>duration</code> or by supplying ending offsets instead. Default Value: FALSE
<code>eof_timeout</code>	A timeslice of type <code>INTERVAL</code> . If a <code>COPY</code> command does not receive any Kafka messages within the <code>eof_timeout</code> interval, Vertica responds by ending that <code>COPY</code> statement. This parameter applies only if <code>stop_on_eof</code> is <code>TRUE</code> .

Loading from Multiple Topics in the Same Stream Parameter

You can load from multiple Kafka topics in a single stream parameter as long as you follow these guidelines:

- The data for the topics must be in the same format because you pass the data from `KafkaSource` to a single parser. For example, you cannot load data from one topic that is in Avro format and another in JSON format.
- Similarly, you need to be careful if you are loading Avro data and specifying an external schema from a registry. The Avro parser accepts a single schema per data load. If the data from the separate topics have different schemas, then all of the data from one of the topics will be rejected.
- The data in the different topics should have the same (or very similar) schemas, especially if you are loading data into a traditional Vertica table. While you can load data with different schemas into a flex table, there are only a few scenarios where it makes sense to combine dissimilar data into a single table.

Duration Note

The `duration` parameter applies to the length of time that Vertica allows the `KafkaSource` function to run. It usually reflects the amount of time the overall load statement takes.

However, if KafkaSource is loading a large volume of data or the data needs extensive processing and parsing, the overall runtime of the query can exceed the amount of time specified in `duration`.

Example

The following example demonstrates calling KafkaSource to load data from Kafka into an existing flex table named `web_test` with the following options:

- The stream is named `web_hits` which has a single partition.
- The load starts at the earliest message in the stream (identified by passing `-2` as the start offset).
- The load ends when it reaches the message with offset `10000`.
- The Kafka cluster's brokers are `kafka01` and `kafka03` in the `example.com` domain.
- The brokers are listening on port `6667`. (This is port number that the Hortonworks Hadoop distribution assigns to its Kafka brokers.)
- The load ends if it reaches the end of the stream before reaching the message with offset `10000`. If you do not supply this option, the connector waits until Kafka sends a message with offset `10000`.
- The loaded data is sent to the `KafkaJSONParser` for processing.

```
=> COPY web_test
      SOURCE KafkaSource(stream='web_hits|0|-2|10000',
                        brokers='kafka01.example.com:6667,kafka03.example.com:6667',
                        stop_on_eof=true)
      PARSER KafkaJSONParser();
Rows Loaded
-----
          1068
(1 row)
```

KafkaOffsets

The `KafkaOffsets` user-defined transform function returns load operation statistics generated by the most recent invocation of `KafkaSource`. Query `KafkaOffsets` to see the metadata produced by your most recent load operation. You can query `KafkaOffsets` after each

KafkaSource invocation to view information about that load. If you are using the scheduler, you can also view historical load information in the [stream_microbatch_history](#) table.

For each load operation, KafkaOffsets returns the following:

- source kafka topic
- source kafka partition
- starting offset
- ending offset
- number of messages loaded
- number of bytes read
- duration of the load operation
- end message
- end reason

The following example demonstrates calling KafkaOffsets to show partition information on the table named `web_test` that was loaded using KafkaSource.

```
=> SELECT kpartition, start_offset, end_offset, msg_count, ending FROM (select KafkaOffsets() over()  
FROM web_test) AS stats ORDER BY kpartition;  
kpartition | start_offset | end_offset | msg_count | ending  
-----+-----+-----+-----+-----  
0 | -2 | 9999 | 1068 | END_OFFSET
```

The output shows that KafkaSource loaded 1068 messages (rows) from Kafka in a single partition. The KafkaSource ended the data load because it reached the ending offset.

Note: The values shown in the `start_offset` column are exclusive (the message with the shown offset was not loaded) and the values in the `end_offset` column are inclusive (the message with the shown offset was loaded). This is the opposite of the values specified in the KafkaSource's `stream` parameter. The difference between the inclusiveness of KafkaSource's and KafkaOffset's start and end offsets are based on the needs of the job scheduler. KafkaOffset is primarily intended for the job scheduler's use, so the start and end offset values are defined so the scheduler can easily start streaming from where left off.

KafkaParser

The KafkaParser does not take any parameters. The parser loads the data bytes into a regular Vertica table directly from Kafka. You can use this parser as a catch-all for unsupported formats.

KafkaJSONParser

The KafkaJSONParser parses JSON-formatted Kafka messages and loads them into a regular Vertica table or a Vertica flex table.

The syntax for calling the parser is:

```
KafkaJSONParser(  
    [flatten_maps=Boolean]  
    [, flatten_arrays=Boolean]  
    [, start_point=Boolean]  
    [, omit_empty_keys=Boolean]  
    [, reject_on_duplicate=Boolean]  
    [, reject_on_materialized_type_error=Boolean]  
    [, reject_on_empty_key=Boolean])
```

Parameter	Description
flatten_maps	Flattens all JSON maps if set to TRUE
flatten_arrays	Flattens JSON arrays if set to TRUE
start_point	Specifies the name of a key in the JSON load data at which to begin parsing. The parser ignores all data before the start_point value. The parser processes data after the first instance, and up to the second, ignoring any remaining data.
omit_empty_keys	If set to TRUE, omits any key from the load data that does not have a value set.
reject_on_duplicate	If set to TRUE, rejects data that contains duplicate key names.
reject_on_materialized_type_error	When set to TRUE, rejects the data row if the data includes keys matching an existing materialized column and has a key that cannot be mapped into the materialized column's data

Parameter	Description
	type.
reject_on_empty_key	If set to TRUE, rejects any row containing a key without a value.

See [Loading JSON Data](#) for more information.

The following example demonstrates loading JSON data from Kafka. The parameters in the statement define to the load to:

- Load data into the pre-existing table named logs.
- The KafkaSource streams the data from a single partition in the source called server_log.
- The Kafka broker for the data load is running on the host named kafka01 on port 9092.
- KafkaSource stops loading data after either 10 seconds or on reaching the end of the stream, whichever happens first.
- The KafkJSONParser flattens any arrays or maps in the JSON data.

```
=> COPY logs SOURCE KafkaSource(stream='server_log|0|0',
                                stop_on_eof=true,
                                duration=interval '10 seconds',
                                brokers='kafka01:9092')
    PARSER KafkaJSONParser(flatten_arrays=True, flatten_maps=True);
```

KafkaAVROParser

The KafkaAVROParser parses AVRO-formatted Kafka messages and loads them into a regular Vertica table or a Vertica flex table.

```
KafkaAVROParser(  
    [reject_on_materialized_type_error=Boolean]  
    [, flatten_maps=Boolean]  
    [, flatten_arrays=Boolean]  
    [, flatten_records=Boolean]  
    [, external_schema=JSON_string]  
    [, codec='default'|'snappy'|'null']  
    [, with_metadata=Boolean]  
    [, schema-registry-url=url]  
    [, schema_registry_subject=subject_name]  
    [, schema_registry_version=version_number])
```

Parameter	Description
<code>reject_on_materialized_type_error</code>	When set to TRUE, rejects the data row if it contains a materialized column value that cannot be mapped into the materialized column's data type.
<code>flatten_maps</code>	If set to TRUE, flattens all Avro maps.
<code>flatten_arrays</code>	If set to TRUE, flattens Avro arrays.
<code>flatten_records</code>	If set to TRUE, flattens all Avro records.
<code>external_schema</code>	Specifies the schema of the Avro file as a JSON string. If this parameter is not specified, the parser assumes that each message has the schema on it. If you are using a schema registry, do not use this parameter.
<code>codec</code>	Specifies the codec in which the Avro file was written. Valid values are: <ul style="list-style-type: none"> 'default' - Avro's default 'snappy' - snappy compression 'null' - data is not compressed and codec is not needed
<code>with_metadata</code>	If set to TRUE, messages include Avro datum, schema, and object metadata. By default, the KafkaAvroParser parses messages without including schema and metadata. If you enable this parameter, write your messages using the Avro API and confirm they contain only Avro datum. The default value is FALSE.
<code>schema_registry_url</code>	Specifies the URL of the Confluent schema registry. This parameter is required to load data based on a schema registry version. If you are using an external schema, do not use this parameter. For more information, refer to Using a Schema Registry with Kafka .
<code>schema_registry_subject</code>	In the schema registry, the subject of the schema to use for data loading.
<code>schema_registry_version</code>	In the schema registry, the version of the schema to use for data loading.

See [Loading Avro Data](#) for more information.

The following example demonstrates loading data from Kafka in an Avro format. The statement:

- Loads data into an existing flex table named `weather_logs`.
- Copies data from the default Kafka broker (running on the local system on port 9092).
- The source is named `temperature`.
- The source has a single partition.
- The load starts from offset 0.
- The load ends either after 10 seconds or the load reaches the end of the source, whichever occurs first.
- The `KafkaAvroParser` does not flatten any arrays, maps, or records it finds in the source.
- The schema for the data is provided in the statement as a JSON string. It defines a record type named `Weather` that contains fields for a station name, a time, and a temperature.
- Rejected rows of data are saved to a table named `t_rejects1`.

```
=> COPY weather_logs
  SOURCE KafkaSource(stream='temperature|0|0', stop_on_eof=true,
                    duration=interval '10 seconds')
  PARSER KafkaAvroParser(flatten_arrays=False, flatten_maps=False, flatten_records=False,
                        external_schema='{"type": "record", "name": "Weather", "fields": '
                                      ' [{"name": "station", "type": "string"}, '
                                      ' {"name": "time", "type": "long"}, '
                                      ' {"name": "temp", "type": "int"} ]}')
  REJECTED DATA AS TABLE "t_rejects1";
```

Parsing Custom Formats

Vertica supports the use of user-defined filters to manipulate data arriving from your streaming message bus. You can apply these filters to data before you parse it. By default, data that flows from the source does not contain message boundaries. The default Kafka parsers can parse this format. However, other user-defined and Vertica parsers require additional message processing. Filters provide the ability to manipulate data using user-defined parsers.

Filters for Use with Kafka Data

Vertica includes the following filters:

- `KafkaInsertDelimiters` — Transforms the Kafka data stream by inserting a user-specified delimiter between each record. The delimiter can contain any characters and be of any length. This parser uses the following syntax:

```
KafkaInsertDelimiters(delimiter = 'delimiter')
```

- `KafkaInsertLengths` — Transforms the Kafka data stream by inserting the length of the following record in bytes at the beginning of the record. Vertica writes lengths as 4-byte uint32 values in Big Endian network byte order. For example, a 100-byte record would be preceded by 0x00000064.

```
KafkaInsertLengths()
```

Note: The Vertica provided filters are mutually exclusive. You cannot use both to process a Kafka data stream.

Vertica also supports the use of additional Vertica and user-defined filters. If you are using a Vertica filter, it must appear first in the filter list. Use a comma to delimit multiple filters. If you are using a non-Kafka parser, you must use at least one filter to prepare your content for that parser. If you do not provide a filter, the parser fails with the message:

```
Input is not from Kafka source.
```

Examples

The following example shows how you can delimit data streams from two hosts by the string `\n`. You can then use a CSV parser to parse the content.

```
=> COPY stream_data.target_table SOURCE KafkaSource (stream='source1|1|1,source2|2|2',  
brokers='host1:9092,host2:9092',  
duration= INTERVAL'timeslice')  
FILTER KafkaInsertDelimiters(delimiter = '\n')  
PARSER MyCsvParser(recordTerminator = '\n');
```

The following example shows how you can specify that a Vertica filter and a decryption filter process a single Kafka data stream. Using the length information the `KafkaInsertLengths` filter injects, the parser can identify each record and parse it individually.

```
=> COPY stream_data.target_table SOURCE KafkaSource (stream='source1|1|1, brokers='host1:9092')  
FILTER KafkaInsertLengths() DecryptFilter(parameter=Key)  
PARSER ComplexParser(parameter = 'value');
```

Using a Schema Registry with Kafka

Vertica supports the use of a Confluent schema registry for Avro schemas with the [KafkaAVROParser](#). By using a schema registry, you enable the Avro parser to parse and decode messages written by the registry and to retrieve schemas stored in the registry. In addition, a schema registry enables Vertica to process streamed data without sending a copy of the schema with each record. Vertica can access a schema registry in the following ways:

- schema ID
- subject and version

Note: If you use the compatibility config resource in your schema registry, you must specify a value of BACKWARD. For more information on installing and configuring a schema registry, refer to the [Confluent documentation](#).

Schema ID Loading

In schema ID based loading, the Avro parser checks the schema ID associated with each message to identify the correct schema to use. A single COPY statement can reference multiple schemas. Because each message is not validated, Vertica recommends that you use a flex table as the target table for schema ID based loading.

The following example shows a COPY statement that refers to a schema registry located on the same host.

```
=> COPY logs source kafkasource(stream='simple|0|0', stop_on_eof=true,  
duration=interval '10 seconds') parser  
KafkaAvroParser(schema_registry_url='http://localhost:8081/');
```

Subject and Version Loading

In subject and version loading, you specify a subject and version in addition to the schema registry URL. The addition of the subject and version identifies a single schema to use for all messages in the COPY. If any message in the statement is incompatible with the schema, the COPY fails. Because all messages are validated prior to loading, Vertica recommends that you use a standard Vertica table as the target for subject and version loading.

The following example shows a COPY statement that identifies a schema subject and schema version as well as a schema registry.

```
=> COPY t source kafkasource(stream='simpleEvolution|0|0',
stop_on_eof=true, duration=interval '10 seconds') parser
KafkaAvroParser(schema_registry_url='http://repository:8081/schema-repo/',
schema_registry_subject='simpleEvolution-value', schema_registry_version='1')
REJECTED DATA AS TABLE "t_rejects";
```

Configuring Kafka for Vertica

The following are settings that are used to optimize Kafka performance with Vertica. Some settings, such as `message.max.bytes`, are configurable on multiple Kafka components.

For detailed information on Apache Kafka configuration settings, refer to the [Apache Kafka rdkafka documentation](#).

Producer Settings

Kafka producers are the processes that publish messages to Kafka brokers.

Setting	Affects	Notes
<code>queue.buffering.max.messages</code>	Latency	Specifies the size of the Vertica producer queue. If Vertica generates too many messages too quickly, the queue can fill, resulting in dropped messages. Increasing this value consumes more memory, but reduces the chance of lost messages.
<code>queue.buffering.max.ms</code>	Latency	Specifies the frequency with which Vertica flushes the producer message queue. Lower values decrease latency at the cost of throughput. Higher values increase throughput, but can cause the producer queue (set by <code>queue.buffering.max.messages</code>) to fill more frequently, resulting in dropped messages.
<code>message.max.bytes</code>	Reliability	Specifies the maximum size of a Kafka message. This size is the size of the JSON serialized message. To prevent truncated messages, set this value to the size of the largest possible message. This values should be the same on your sources, brokers, and producers.
<code>message.send.max.retries</code>	Reliability	Specifies the number of attempts the

Setting	Affects	Notes
		producer makes to deliver the message to a broker. Higher values increase the chance of success.
retry.backoff.ms	Reliability	Specifies the interval Vertica waits before resending a failed message.
request.required.acks	Reliability	Specifies how many broker replica acknowledgments Kafka requires before it considers message delivery successful. Requiring acknowledgments increases latency. Removing acknowledgments increases the risk of message loss.
request.timeout.ms	Reliability	Specifies the interval that the producer waits for a response from the broker. Broker response time is affected by server load and the number of message acknowledgments you require. Higher values increase latency.

Broker Settings

Kafka brokers receive messages from producers and distribute them among Kafka consumers. Configure these settings on the brokers themselves. These settings function independently of your producer and consumer settings.

Setting	Affects	Notes
message.max.bytes	Reliability	Specifies the maximum size of a Kafka message. This size is the size of the JSON serialized message. To prevent truncated messages, set this value to the size of the largest possible message. This values should be the same on your sources, brokers, and producers.
num.network.threads	Performance	Specifies the number of network threads the broker uses to accept network requests. More threads can increase your concurrency.

Setting	Affects	Notes
num.io.threads	Performance	Specifies the number of network threads the broker uses to receive and process requests. More threads can increase your concurrency.

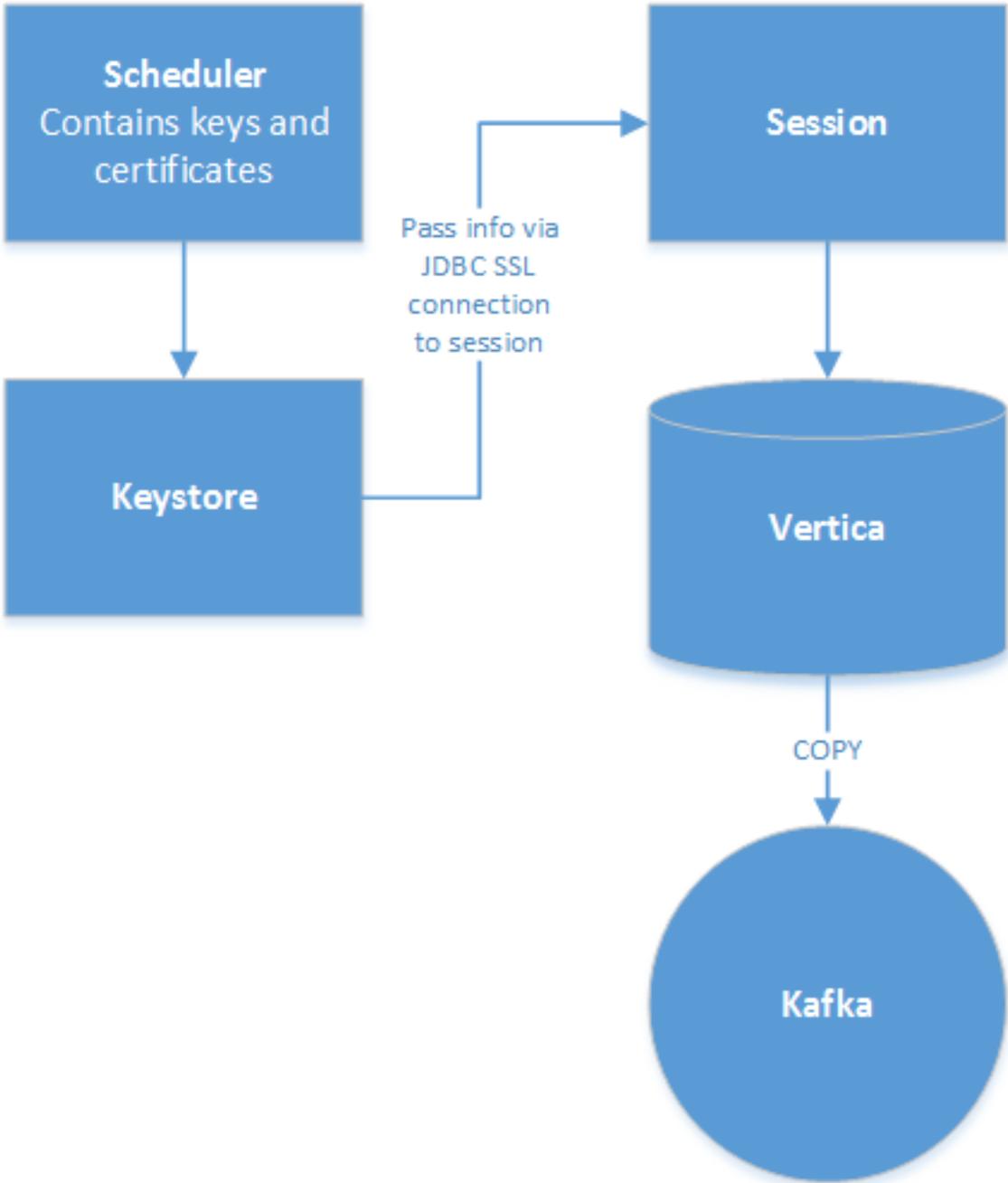
Consumer Settings

Kafka consumers process the stream of messages distributed by the brokers. You can set these values using the `kafka_conf` parameter on the KafkaSource UDL or using the `--parser-parameters` setting in the [Load Spec Utility Options](#).

Setting	Affects	Notes
message.max.bytes	Reliability	Specifies the maximum size of a Kafka message. This size is the size of the JSON serialized message. To prevent truncated messages, set this value to the size of the largest possible message. This values should be the same on your sources, brokers, and producers.

Using SSL with Kafka

Vertica supports the use of SSL authentication between Kafka, Vertica, and the Kafka Scheduler. For information on SSL authentication with Vertica, refer to [TLS/SSL Server Authentication](#).



Scheduler/Vertica Communication

The scheduler uses keystore information on a per-session basis to communicate with Vertica. It passes information through a JDBC SSL connection to Vertica. The Scheduler connects to Vertica through a JVM. As a result, your SSL keys must be within the JVM keystore and the CA must be within the JVM truststore. When connecting between Vertica and Kafka, the Scheduler uses its keys to authenticate Kafka.

Vertica/Kafka Communication

In a typical configuration, each Kafka broker contains its own key store and trust store. Vertica and Kafka authenticate against your certifying authority. Kafka authenticates by means of the librdkafka library.

Adding SSL to Vertica/Kafka Integration

To add SSL authentication to your Vertica/Kafka integration, you must configure SSL support on various components in Vertica, Kafka, and Java.

1. [Configure Vertica for SSL.](#)
2. [Configure Kafka for SSL.](#)
3. [Configure your scheduler for SSL.](#)
4. [Configure Java on your scheduler for SSL.](#)

Configuring Kafka for SSL

A typical Kafka/Vertica configuration consists of the following steps:

1. [Create a certifying authority certificate.](#)
2. [Create a trust store and key store on your Kafka brokers using your certificate.](#)

3. [Configure the server.properties file on each Kafka broker to use your key store.](#)

Create a Certifying Authority Certificate

If you do not already have one, create a certifying authority certificate, as shown in the following example.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

For more information on generating a certificate, refer to the [OpenSSL documentation](#).

Create a Trust Store and Key Store on your Kafka Brokers

You must create a trust store and key store on each of your Kafka brokers.

1. Create a trust store on your Kafka broker by importing your certificate. The following example shows a typical trust store command.

```
keytool -keystore kafka.truststore.jks -alias caroot -import -file ca-cert
```

2. Create a key store on your Kafka broker, as shown in the following example. The following example shows a typical series of key store commands.

```
keytool -keystore brokername.keystore.jks -alias brokername -validity 365 -keyalg RSA -genkey  
keytool -keystore brokername.keystore.jks -alias brokername -certreq -file cert-file  
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -  
CAcreateserial  
keytool -keystore brokername -alias caroot -import -file ca-cert  
keytool -keystore brokername.keystore.jks -alias broker$$i -import -file cert-signed
```

3. Repeat the key store and trust store configuration process for each Kafka broker, specifying the correct alias and key store name for that broker.

Configure the server.properties File on Each Kafka Broker to Use Your Key Store

You can pass SSL configuration information from your Kafka broker to Vertica by including [user defined session parameters](#) in your server.properties file. For more information on configuring your server.properties file, refer the [Apache Kafka documentation](#).

```
kafka_Enable_SSL=1  
kafka_SSL_CA=<Certifying Authority contents>  
kafka_SSL_Certificate=<Certificate contents>  
kafka_SSL_PrivateKey_secret=<Private Key contents>  
kafka_SSL_PrivateKeyPassword_secret=<Private Key password>
```

Configuring Your Scheduler for SSL

Your scheduler requires a trust store, key store and launch parameters to use SSL.

Create a Trust Store and and Key Store on Your Scheduler

You must create a trust store and key store on your scheduler.

1. Create a trust store on your scheduler by importing your certificate. The following example shows a typical trust store command.

```
keytool -keystore schedulername.truststore.jks -alias caroot -import -file ca-cert
```

2. Create a key store on your scheduler, as shown in the following example. The following example shows a typical series of key store commands.

```
keytool -keystore schedulername.keystore.jks -alias vsched -validity 365 -keyalg RSA -genkey  
keytool -keystore schedulername.keystore.jks -alias vsched -certreq -file cert-file  
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365  
keytool -keystore schedulername.keystore.jks -alias caroot -import -file ca-cert  
keytool -keystore schedulername.keystore.jks -alias vsched -import -file cert-signed
```

Launch Your Scheduler with SSL Enabled

When you launch your scheduler, you must include SSL parameters to enable SSL support. For more information on scheduler utility parameters, refer to [Scheduler Utility Options](#).

The following example shows a launch command including all the required SSL parameters

```
/opt/vertica/packages/kafka/bin/vkconfig launch --enable-SSL true  
--ssl-ca-alias authenticcert --ssl-key-alias ourkey --ssl-key-password secret
```

Configuring Java to Use SSL

The scheduler uses JDBC to communicate with your Vertica session. As a result, you must configure your scheduler JVM and JDBC settings for SSL. For information on configuring your JVM and JDBC, refer to the [Java documentation](#).

Configure Your Scheduler JVM

To use SSL authentication with Kafka, you must configure your scheduler JVM to use the key store and trust store you created. Vertica recommends that you use a configuration file to provide this information. Include the following options in your configuration file:

```
-DJVMjavax.net.ssl.trustStore=/path/to/truststore  
-DJVMjavax.net.ssl.keyStore=/path/to/keystore  
-DJVMjavax.net.ssl.keyStorePassword=password1234
```

Configure Your Scheduler JDBC to use SSL

Add the following to your generated JDBC string:

```
--jdbc-opt ssl=true
```

Using SSL Without a Scheduler

You can also configure Vertica to interact with Kafka without using a scheduler. In this case, Kafka integration requires user-defined extensions (UDXs) that do not have access to global Vertica configuration parameters. As a result, you must load your SSL Certificates as session parameters.

1. Configure your [MaxSessionUDParameterSize](#) to a value larger than the length of your certificate chain.

```
ALTER SESSION SET MaxSessionUDParameterSize=100000
```

2. Provide certificate, key, and password information to your session. Vertica does not log parameters that end in `_secret`. For more information on these parameters, refer to [User-Defined Session Parameters](#).

```
=> ALTER SESSION SET UDPARAMETER kafka_SSL_Certificate='<client.crt contents>';  
ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKey_secret='<client.key contents>';  
ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKeyPassword_secret='<password, if applicable>';  
ALTER SESSION SET UDPARAMETER kafka_SSL_CA='<ca.crt contents>';
```

3. Enable SSL authentication for your session.

```
=> ALTER SESSION SET kafka_Enable_SSL=1;
```

You can now run COPY commands with SSL authentication.

Streaming Utility Options

Vertica includes the `vkconfig` utility to help you configure your streaming data resources for use with your database.

The streaming data integration feature for Vertica consists of the following utilities:

- [Shared Utility Options](#)
- [Cluster Utility Options](#)
- [Scheduler Utility Options](#)
- [Load Spec Utility Options](#)
- [Microbatch Utility Options](#)
- [Source Utility Options](#)
- [Target Utility Options](#)
- [Launch Utility Options](#)

You can use the options in the [Shared Utility Options](#) table with any of the utilities. Utility-specific options appear in their respective tables.

Shared Utility Options

These utility options are available across the functional areas of the streaming data configuration utility.

Option	Description
<code>--config-schema</code> <i>schema_name</i>	Identifier for the Vertica schema. This value is the same name as the scheduler. You use this name to identify the scheduler during configuration. Default Value: stream_config
<code>--help</code>	Prints out a help menu listing available options with a description.

Option	Description
<code>--version</code>	Displays the version number of the scheduler.
<code>--dbhost <i>host name</i></code>	The host name or IP address of the Vertica node acting as the initiator node for the scheduler. Default Value: localhost
<code>--dbport <i>port_number</i></code>	The port to use to connect to a Vertica database. Default Value: 5433
<code>--username <i>username</i></code>	The Vertica database user. Default Value: Current user
<code>--password <i>password</i></code>	Password for the database user.
<code>--jdbc-url <i>url</i></code>	A complete JDBC URL that overrides other database connection parameters.
<code>--conf <i>filename</i></code>	A properties file (.properties) that contains configuration details for the CLI command. The configuration file follows the standard java properties file format. Using a properties file is helpful for options that do not change across CLI commands. Any additional options added at the command line override parameters set in the properties file. The configuration file supports all Kafka properties. For a list of properties, refer to kafka.apache.org/08/configuration.html .
<code>--create</code>	Creates a new instance of the supplied type. Renamed from <code>--add</code> .
<code>--update</code>	Updates an existing instance of the supplied type. Renamed from <code>--edit</code> .
<code>--delete</code>	Delete an instance of the supplied type. Renamed from <code>--remove</code> .

Examples

These examples show how you can use the shared utility options.

Display help for the scheduler utility:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --help
PARAMETER      REQUIRED #ARGS  DESCRIPTION
conf            no          1          Allow the use of a properties file to associate parameter keys and
values. This file enables command string reuse and cleaner command strings.
help           no          0          Outputs a help context for the given subutility.
version        no          0          Outputs the current Version of the scheduler.
```

Use a property file to store parameters for the utility. You can override any stored parameter from the command line:

```
#config.properties:
username=myuser
password=mypassword
dbhost=localhost
dbport=5433

/opt/vertica/packages/kafka/bin/vkconfig source --update --conf config.properties
```

Scheduler Utility Options

The scheduler is a tool that continuously loads data from Kafka into Vertica. Use the scheduler utility to create, update, or delete a scheduler, defined by `config-schema`. If you do not specify a scheduler, utility commands apply to the default `stream_config` scheduler.

Option	Description
<code>--operator <i>username</i></code>	<p>Allows the dbadmin to grant privileges to a previously created Vertica user.</p> <p>This option gives the specified user all privileges on the scheduler instance and EXECUTE privileges on the libkafka library and all its UDxs.</p> <p>Granting operator privileges gives the user the right to read data off any source in any cluster that can be reached from the Vertica node.</p>

Option	Description
	<p>The dbadmin must grant the user separate permission for them to have write privileges on the target tables.</p> <p>Requires the <code>--create shared utility</code> option.</p> <p>To revoke privileges, use the <code>--remove</code> option with the <code>--operator</code> option.</p>
<code>--drop schema_name</code>	Drops the specified schema. After you drop the configuration information, you cannot recover it.
<code>--add-operator user_name</code>	Adds a user account that operates the specified configuration. Requires the <code>--update shared utility</code> option.
<code>--remove-operator user_name</code>	Deletes a user account that operates the specified configuration. Requires the <code>--update shared utility</code> option.
<code>--upgrade</code>	Upgrades the existing scheduler and configuration schema to the current version. The upgraded version of the scheduler is not backwards compatible with earlier versions. To upgrade an alternate schema, use the <code>upgrade-to-schema</code> parameter.
<code>--upgrade-to-schema schema_name</code>	Specifies a configuration schema to use with the upgraded scheduler. Requires the <code>--upgrade scheduler utility</code> option. If you do not include this parameter with an upgrade, Vertica upgrades the current schema.
<code>--frame-duration HH:MM:SS</code>	<p>The interval of time that all individual frames last with this scheduler. Vertica must have enough time to complete COPY tasks within this duration. You can calculate the average available time per COPY using the following equation:</p> $TimePerCopy = (FrameDuration * Parallelism) / Microbatches$ <p>Vertica requires at least 100 milliseconds per COPY to function. You can increase the available time per COPY by increasing your frame duration.</p> <p>Default Value:</p> <p>00:00:10</p>
<code>--config-refresh HH:MM:SS</code>	The interval of time that the scheduler runs before synchronizing its settings and updating its cached metadata (such as changes made

Option	Description
	<p>by using the <code>--update</code> option). Renamed from <code>--config-refresh-interval</code>.</p> <p>Default Value: 00:05:00</p>
<p><code>--resource-pool</code> <i>pool_name</i></p>	<p>The resource pool to be used by all queries executed by this scheduler. You must create this pool in advance if you are not using the default pool.</p> <p>Default Value: stream_default_pool</p>
<p><code>--new-source-policy</code> FAIR START END</p>	<p>Determines how much Vertica allocates resources to the newly added source.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> • FAIR: Takes the average length of time from the previous batches and schedules itself appropriately. • START: All new sources start at the beginning of the frame. The batch receives the minimal amount of time to run. • END: All new sources start at the end of the frame. The batch receives the maximum amount of time to run. <p>Default Value: FAIR</p>
<p><code>--eof-timeout-ms</code> <i>number of milliseconds</i></p>	<p>If a COPY command does not receive any messages within the eof-timeout-ms interval, Vertica responds by ending that COPY statement.</p> <p>Default Value: 1 second</p> <p>See Using COPY with Data Streaming in this guide for more information.</p>
<p><code>--message_max_bytes</code> <i>max_message_size</i></p>	<p>The maximum message size, in bytes.</p> <p>Default Value:</p>

Option	Description
	1048576
<code>--fix-config</code>	Repairs the configuration and re-creates any missing tables. Valid only with the <code>--update</code> shared configuration option.
<code>--validation-type</code> ERROR WARN SKIP	<p>Specifies the level of validation performed on the scheduler. Invalid SQL syntax and other errors can cause invalid micro-batches. Vertica supports the following validation types:</p> <ul style="list-style-type: none"> • ERROR - Cancel configuration or creation if validation fails. If you do not specify a validation type, this value is the default. • WARN - Proceed with task if validation fails, but display a warning. • SKIP - Perform no validation. <p>For more information on validation, refer to Data Streaming Job Scheduler.</p> <p>Renamed from <code>--skip-validation</code>.</p>
<code>--auto-sync</code> TRUE FALSE	<p>When TRUE, Vertica automatically synchronizes scheduler source information at the interval specified in <code>--config-refresh</code>.</p> <p>Default Value:</p> <p>TRUE</p> <p>For more information on synchronization, refer to Data Streaming Job Scheduler.</p>

Examples

These examples show how you can use the scheduler utility options.

Give a user, Jim, privileges on the StreamConfig scheduler. Specify that you are making edits to the StreamConfig scheduler with the `--config-schema` option:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --update --config-schema StreamConfig --add-operator Jim
```

Edit the default `stream_config` scheduler so that every COPY statement waits for data for one second before ending:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --update --eof-timeout-ms 1000
```

Upgrade the scheduler and streaming schema to the current version:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --upgrade --config-schema old_config --upgrade-to-schema new_schema
```

Drop the schema scheduler219a:

```
/opt/vertica/packages/kafka/bin/vkconfig scheduler --drop --config-schema scheduler219a --username release
```

Cluster Utility Options

The Kafka Cluster utility enables you, as an administrator, to connect multiple clusters to a single Vertica database.

Option	Description
<code>--cluster</code> <i>cluster_name</i>	A unique, case insensitive name for the cluster.
<code>--new-cluster</code> <i>cluster_name</i>	The updated name for the cluster. Requires the <code>--update</code> shared utility option.
<code>--hosts</code> <i>b1:port,b2:port...</i>	Identifies the hosts that you want to add, edit, or remove from a Kafka cluster. To identify multiple hosts, use a comma delimiter. Renamed from <code>--brokers</code> .

Note: Vertica does not validate host lists or confirm that clusters are unique. Labeling the same cluster with multiple IDs can introduce duplicate data into your database.

Examples

This example shows how you can create the cluster, StreamCluster1, and assign two hosts:

```
/opt/vertica/packages/kafka/bin/vkconfig cluster --create --cluster StreamCluster1 --hosts 10.10.10.10:9092,10.10.10.11:9092
```

Source Utility Options

Use the source utility to create, update, or delete a source.

Option	Description
<code>--source</code> <i>source_name</i>	The source to add, remove, or edit from a scheduler configuration. In most cases, the source is the name of your Kafka cluster.
<code>--new-source</code> <i>source_name</i>	Updates the name of an existing source to the name specified by this parameter. Requires: <code>--update</code> shared utility option
<code>--partitions</code> <i>count</i>	Used to create all partitions from 0 to $n-1$. Default Value: 1 Requires: <code>--create</code> and <code>--source</code> options You must keep this consistent with the number of partitions in the source. Renamed from <code>--num-partitions</code> .
<code>--cluster</code> <i>cluster_name</i>	Identifies the cluster that you want to edit.
<code>--new-cluster</code> <i>cluster_name</i>	Updates the name of an existing cluster to the name specified by this parameter. All sources referencing the old cluster source now target this cluster. Requires: <code>--update</code> and <code>--source</code> options
<i>enabled</i> TRUE FALSE	When TRUE, the source is available for use.

Examples

The following examples show how you can create or update SourceFeed and assign it to a cluster.

Create the source SourceFeed and assign it to the cluster, StreamCluster1:

```
/opt/vertica/packages/kafka/bin/vkconfig source --create --source SourceFeed --cluster StreamCluster1 --partitions 3
```

Update the existing source SourceFeed to use the existing cluster, StreamCluster2:

```
/opt/vertica/packages/kafka/bin/vkconfig source --update --source SourceFeed --new-cluster StreamCluster2
```

Target Utility Options

Use the target utility to configure a Vertica table to receive data from your streaming data application.

Option	Description
<code>--target-schema</code> <i>schema.table_name</i>	The existing Vertica target schema associated with this target.
<code>--new-target-schema</code> <i>schema_name</i>	Changes the Vertica schema associated with this schema to a new, already created schema. Requires: <code>--update</code> option.
<code>--target-table</code> <i>table_name</i>	The name of a Vertica table corresponding to the target.
<code>--new-target-table</code> <i>schema_name</i>	Changes the Vertica target table associated with this schema to a new, already created table. Requires: <code>--update</code> option.

Important: Avoid having columns with NOT NULL restrictions in your target table. The scheduler stops loading data if it encounters a row that has a NULL value it needs to insert into a column with a NOT NULL restriction. If you must have a NOT NULL column, try to filter out all NULL values for that column in the streamed data before it is loaded by the scheduler.

Examples

This example shows how you can create a target from public.streamtarget:

```
/opt/vertica/packages/kafka/bin/vkconfig target --create --target-schema public --target-table streamtarget
```

Load Spec Utility Options

The Load Spec utility enables you to provide parameters for the COPY statement that loads streaming data.

Option	Description
<code>--load-spec <i>spec_name</i></code>	A unique name for this copy load spec.
<code>--filters "<i>filter_name</i>"</code>	A Vertica FILTER chain containing all UDFilters for the COPY statement. For more information on filters, refer to Parsing Custom Formats .
<code>--message-max-bytes <i>max_message_size</i></code>	Specifies the maximum size, in bytes, of a message. Default Value: 1048576
<code>--parser <i>parser_name</i></code>	Identifies a VerticaUDParser to use with a specified target. If you are using a Vertica native parser, parser parameters serve as a COPY statement parameters. Default Value: KafkaParser
<code>--parser-parameters <i>key=value, key=value</i></code>	A list of parameters to provide to the parser.
<code>--new-load-spec</code>	A new, unique name for an existing load spec. Requires the <code>--update</code> parameter
<code>--uds-kv-parameters <i>key=value, key=value</i></code>	A comma separated list of key value pairs for the user-defined source.

Option	Description
<code>--load-method</code> AUTO TRICKLE DIRECT	The COPY load method to use for all loads with this scheduler. See the COPY statement for more information. Default Value: TRICKLE

Examples

These examples show how you can use the Load Spec utility options.

Create the load spec, Streamspec1:

```
/opt/vertica/packages/kafka/bin/vkconfig load-spec --create --load-spec Streamspec1
```

Update the load spec, Streamspec1, to the name Streamspec2, and also update the load method to Direct:

```
/opt/vertica/packages/kafka/bin/vkconfig load-spec --update --load-spec Streamspec1 --new-load-spec Streamspec2 --load-method direct
```

Update the load spec, Filterspec, to use the KafkaInsertLengths filter and a custom decryption filter:

```
/opt/vertica/packages/kafka/bin/vkconfig load-spec --update --load-spec Filterspec --filters "KafkaInsertLengths() DecryptFilter(parameter=Key)"
```

Microbatch Utility Options

A microbatch represents an individual segment of a data load from a Kafka stream. It combines the definitions for your cluster, source, target, and load spec that you create using the other vkconfig options. The scheduler uses all of the information in the microbatch to execute [COPY](#) statements using [KafkaSource](#) function that transfer data from Kafka to Vertica. The result of each microbatch load is stored in the [stream_microbatch_history](#) table.

Option	Description
<code>--microbatch</code> <i>name</i>	A unique, case insensitive name for the microbatch.

Option	Description
<code>--new-microbatch</code> <i>updated_name</i>	The updated name for the microbatch. Requires the <code>--update</code> shared utility option.
<code>--load-spec</code> <i>loadspect_name</i>	The load spec to use while processing this microbatch.
<code>--target-schema</code> <i>schema_name</i>	The existing Vertica target schema associated with this microbatch.
<code>--rejection-schema</code> <i>schema_name</i>	The existing Vertica schema that contains a table for storing rejected messages.
<code>--target-columns</code> <i>column_name, ... column expression</i>	A column expression for the target table. This value can be a list of columns or a complete expression. See the COPY statement COPY Parameters in the core documentation for a description of column expressions.
<code>--rejection-table</code> <i>table_name</i>	The existing Vertica table that stores rejected messages.
<code>--enabled</code> TRUE FALSE	When TRUE, allows the microbatch to execute.
<code>--target-table</code> <i>table_name</i>	The name of a Vertica table corresponding to the target. This table must belong to the target schema.
<code>--add-source</code> <i>source_name</i>	The name of a source to assign to this microbatch. You can use this parameter once per command. You can also use it with <code>--update</code> to add sources to a microbatch. Requires <code>--add-source-cluster</code> .
<code>--add-source-cluster</code> <i>cluster_name</i>	The name of a cluster to assign to this microbatch. You can use this parameter once per command. You can also use it with <code>--update</code> to add sources to a microbatch. You can only add sources from the same cluster to a single microbatch. Requires <code>--add-source</code> .
<code>--remove-source</code> <i>source_name</i>	The name of a source to remove from this microbatch. You can use this parameter once per command. You can also use it with <code>--update</code> to

Option	Description
<i>name</i>	remove multiple sources from a microbatch. Requires <code>--remove-source-cluster</code> .
<code>--remove-source-cluster</code> <i>cluster_name</i>	The name of a cluster to remove from this microbatch. You can use this parameter once per command. Requires <code>--remove-source</code> .
<code>--offset</code> <i>partition_1_offset</i> [, <i>partition_2_offset</i> ,...]	<p>The offset of the message in the source where the microbatch starts its load. If you use this parameter, you must supply an offset value for each partition in the source or each partition you list in the <code>--partition</code> option.</p> <p>You can use this option to skip some messages in the source or reload previously read messages.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Important: You cannot set an offset for a microbatch while the scheduler is running. If you attempt to do so, the <code>vkconfig</code> utility returns an error. Use the <code>shutdown</code> utility to shut the scheduler down before setting an offset for a microbatch.</p> </div>
<code>--partition</code> <i>partition_1</i> [, <i>partition_2</i> ,...]	One or more partitions to which the offsets given in the <code>--offset</code> option apply. If you supply this option, then the offset values given in the <code>--offset</code> option applies to the partitions you specify. Requires the <code>--offset</code> option.
<code>--source</code> <i>source_name</i>	The name of the source to which the offset in the <code>--offset</code> option applies. Required when the microbatch defines more than one source or the <code>--cluster</code> parameter is given. Requires the <code>--offset</code> option.
<code>--cluster</code> <i>cluster_name</i>	The name of the cluster to which the <code>--offset</code> option applies. Only required if the microbatch defines more than one cluster or the <code>--source</code> parameter is supplied. Requires the <code>--offset</code> option.

Examples

This example shows how you can create the microbatch, `mbatch1`. This microbatch identifies the schema, target table, load spec, and source for the microbatch:

```
/opt/vertica/packages/kafka/bin/vkconfig microbatch --create --microbatch mbatch1 \  
--target-schema public \  
--target-table BatchTarget \  
--load-spec Filterspec \  
--add-source SourceFeed \  
--add-source-cluster StreamCluster1
```

Launch Utility Options

Use the Vertica launch utility to assign a name to a scheduler instance.

Option	Description
<code>--enable-ssltrue false</code>	(Optional) Enables SSL authentication between Kafka, Vertica, and the Kafka Scheduler. For more information, refer to Using SSL with Kafka .
<code>--ssl-ca-alias</code>	The user-defined alias of the root certifying authority you are using to authenticate communication between Vertica and Kafka. This parameter is used only when SSL is enabled.
<code>--ssl-key-alias</code>	The user-defined alias of the key/certificate pair you are using to authenticate communication between Vertica and Kafka. This parameter is used only when SSL is enabled.
<code>--ssl-key-password</code>	The password used to create your SSL key. This parameter is used only when SSL is enabled.
<code>--instance-name <i>name</i></code>	(Optional) Allows you to name the process running the scheduler. You can use this command when viewing the <code>scheduler_history</code> table, to find which instance is currently running.

Examples

This example shows how you can name the following scheduler instance. Specify the name as PrimaryScheduler:

```
/opt/vertica/packages/kafka/bin/vkconfig launch --instance-name PrimaryScheduler
```

This example shows how you can launch an instance named SecureScheduler with SSL enabled:

```
/opt/vertica/packages/kafka/bin/vkconfig launch --instance-name SecureScheduler --enable-SSL true  
--ssl-ca-alias authenticcert --ssl-key-alias ourkey --ssl-key-password secret
```

Shutdown Utility Options

Use the shutdown utility to terminate all Vertica schedulers. Run this command before restarting the scheduler.

Note: Restarting the scheduler without terminating existing schedulers can produce unexpected behavior.

Examples

This example shows how you can terminate all Vertica Kafka schedulers:

```
/opt/vertica/packages/kafka/bin/vkconfig shutdown
```

Kafka Function Reference

- [KafkaExport](#)

KafkaExport

Outputs rows that Vertica was unable to send to Kafka. If all of your messages have imported successfully, this function returns zero rows. You can use this function to copy failed messages to a secondary table for evaluation and reprocessing.

Syntax

```
KafkaExport(partitionColumn, keyColumn, valueColumn USING PARAMETERS  
brokers='host', source='sourcename')  
OVER () FROM table
```

Parameters

Argument	Description
<code>partitionColumn</code>	Optional. The target partition for the export. If you do not specify a partition, Vertica uses the default partition. You can use the partition column to send messages to partitions that map to Vertica segments. Specify a value of NULL to skip this parameter.
<code>keyColumn</code>	Optional. The user defined key value associated with the <code>valueColumn</code> . Specify a value of NULL to skip this parameter.
<code>valueColumn</code>	Optional. The message itself. The column must have a data type of VARCHAR. Specify a value of NULL to skip this parameter.
<code>brokers</code>	A list of one or more hosts.
<code>source</code>	The source to which you are exporting.

Examples

```
=> SELECT KafkaExport(partion, messageId, message USING PARAMETERS brokers='KafkaBroker'  
, source='failure_test',  
                    kafka_conf='message.max.bytes=64000')  
   OVER (PARTITION BEST)  
   FROM failure_test;  
partition | key | substr | failure_reason  
-----+-----+-----+-----  
    -123 | key1 | negative partition not allowed | Local: Unknown partition  
   54321 |     | nonexistant partition | Local: Unknown partition  
      0 | normal key1 | normal value1 | Broker: Message size too  
large  
      0 |     | aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa | Broker: Message size too  
large  
      0 | normal key2 | normal value2 | Broker: Message size too  
large
```

Data Streaming Schema Tables

Every time you create a scheduler (`--create`), Vertica creates a schema for that scheduler with the name you specify or the default `stream_config`. Each schema has the following tables:

- [stream_clusters](#)
- [stream_events](#)
- [stream_load_specs](#)
- [stream_microbatch_history](#)
- [stream_microbatch_source_map](#)
- [stream_microbatches](#)
- [stream_scheduler](#)
- [stream_scheduler_history](#)
- [stream_sources](#)
- [stream_targets](#)

Caution: Micro Focus recommends that you do not alter these tables except in consultation with support.

stream_clusters

This table lists clusters and hosts.

Column	Data Type	Description
id	INT	The identification number assigned to the cluster.
cluster	VARCHAR	The name of the cluster.
hosts	VARCHAR	A comma-separated list of hosts associated with the cluster.

Examples

This example shows a cluster and its associated hosts.

```
=> SELECT * FROM stream_config.stream_clusters;
  id | cluster | hosts
-----+-----+-----
2250001 | streamcluster1 | 10.10.10.10:9092,10.10.10.11:9092
(1 rows)
```

stream_events

This table logs micro-batches and other important events from the scheduler in an internal log table.

This table was renamed from kafka_config.kafka_events.

Column	Data Type	Description
event_time	TIMESTAMP	The time the event was logged.
log_level	VARCHAR	The type of event that was logged. Valid Values: <ul style="list-style-type: none">TRACEDEBUGFATALERRORWARNINFO Default value: INFO
frame_start	TIMESTAMP	The time when the frame executed.
frame_end	TIMESTAMP	The time when the frame completed.

Column	Data Type	Description
microbatch	INTEGER	The identification number of the associated microbatch.
message	VARCHAR	A description of the event.
exception	VARCHAR	If this log is in the form of a stack trace, this column lists the exception.

Examples

This example shows typical rows from the `stream_events` table.

```
=> SELECT * FROM stream_config.stream_events;

-[ RECORD 1 ]-+-----
event_time   | 2016-07-17 13:28:35.548-04
log_level    | INFO
frame_start  |
frame_end    |
microbatch   |
message      | New leader registered for schema stream_config. New ID: 0, new Host: 10.20.100.62
              | 004:9092,eng-g9-005:9092), resource pool: kafka_default_pool
exception    |
-[ RECORD 2 ]-+-----
event_time   | 2016-07-17 13:28:45.643-04
log_level    | INFO
frame_start  | 2015-07-17 12:28:45.633
frame_end    | 2015-07-17 13:28:50.701-04
microbatch   |
message      | Generated tuples: test3|2|-2,test3|1|-2,test3|0|-2
exception    |
-[ RECORD 3 ]-+-----
event_time   | 2016-07-17 14:28:50.701-04
log_level    | INFO
frame_start  | 2016-07-17 13:28:45.633
frame_end    | 2016-07-17 14:28:50.701-04
microbatch   |
message      | Total rows inserted: 0
exception    |
```

stream_load_specs

This table describes user-created load specs.

Column	Data Type	Description
id	INT	The identification number assigned to the cluster.
load_spec	VARCHAR	The name of the load spec.
filters	VARCHAR	A comma-separated list of hosts associated with the cluster.
parser	VARCHAR	A VerticaUDParser to use with a specified target. If you are using a Vertica native parser, parser parameters serve as a COPY statement parameters.
parser_parameters	VARCHAR	A list of parameters to provide to the parser.
load_method	VARCHAR	The COPY load method to use for all loads with this scheduler. See the COPY statement for more information.
message_max_bytes	INT	The maximum size, in bytes, of a message.
uds_kv_parameters	VARCHAR	A list of parameters provided to the copy statement.

Examples

This example shows the load specs that you can use with a Vertica instance.

```
SELECT * FROM stream_config.stream_load_specs;
-[ RECORD 1 ]-----+-----
id                | 1
load_spec         | loadspec2
filters           |
parser            | KafkaParser
parser_parameters |
load_method       | direct
message_max_bytes| 1048576
uds_kv_parameters |
-[ RECORD 2 ]-----+-----
id                | 750001
load_spec         | streamspec1
filters           |
parser            | KafkaParser
parser_parameters |
load_method       | TRICKLE
message_max_bytes| 1048576
uds_kv_parameters |
```

stream_microbatch_history

This table contains a history of every microbatch executed within this scheduler configuration.

Column	Data Type	Description
source_name	VARCHAR	The name of the source.
source_cluster	VARCHAR	The name of the source cluster. To query sources, refer to stream_sources .
source_partition	INTEGER	The number of the data streaming partition.
start_offset	INTEGER	The starting offset of the microbatch.
end_offset	INTEGER	The ending offset of the microbatch.
end_reason	VARCHAR	An explanation for why the batch ended. The following are valid end reasons: <ul style="list-style-type: none">• DEADLINE - The batch ran out of time• END_OFFSET - The load reached the ending offset specified in the KafkaSource• END_OF_STREAM - There are no messages available to the scheduler or the eof_timeout has been reached• NETWORK_ERROR - The scheduler could not connect to Kafka• RESET_OFFSET - The start offset was changed using the --update and --offset parameters to the KafkaSource. This state does not occur during normal scheduler operations.• SOURCE_ERROR - The specified Kafka topic does not exist• UNKNOWN - The batch ended for an unknown reason
end_reason_	VARCHAR	If the end reason is a network or source issue, this column

Column	Data Type	Description
message		contains a brief description of the issue.
partition_ bytes	INTEGER	The number of bytes transferred from a source partition to a Vertica target table.
partition_ messages	INTEGER	The number of messages transferred from a source partition to a Vertica target table.
microbatch_id	INTEGER	The Vertica transaction id for the batch session.
microbatch	VARCHAR	The name of the microbatch.
target_schema	VARCHAR	The name of the target schema.
target_table	VARCHAR	The name of the target table.
timeslice	INTERVAL	The amount of time spent in the KafkaSource operator.
batch_start	TIMESTAMP	The time the batch executed.
batch_end	TIMESTAMP	The time the batch completed.
last_batch_ duration	INTERVAL	The length of time required to run the complete COPY statement.
consecutive_ error_count	INTEGER	The number of times a microbatch has encountered an error on an attempt to load. This value increases over multiple attempts.
transaction_id	INTEGER	The identifier for the transaction within the session.
frame_start	TIMESTAMP	The time the frame started. A frame can contain multiple microbatches.
frame_end	TIMESTAMP	The time the frame completed.

Examples

This example shows typical rows from the stream_microbatch_history table.

```
=> SELECT * FROM stream_config.stream_microbatch_history;
```

```
-[ RECORD 1 ]--+-+-----
```

```

source_name      | streamsource1
source_cluster  | kafka-1
source_partition | 0
start_offset    | 196
end_offset      | 196
end_reason      | END_OF_STREAM
partition_bytes | 0
partition_messages | 0
microbatch_id   | 1
microbatch      | mb_0
target_schema   | public
target_table    | kafka_flex_0
timeslice       | 00:00:09.892
batch_start     | 2016-07-28 11:31:25.854221
batch_end       | 2016-07-28 11:31:26.357942
last_batch_duration | 00:00:00.379826
consecutive_error_count |
transaction_id  | 45035996275130064
frame_start     | 2016-07-28 11:31:25.751
frame_end       |
end_reason_message |

```

```

-[ RECORD 2 ]--+-----
source_name      | streamsource1
source_cluster  | kafka-1
source_partition | 1
start_offset    | 197
end_offset      | 197
end_reason      | NETWORK_ISSUE
partition_bytes | 0
partition_messages | 0
microbatch_id   | 1
microbatch      | mb_0
target_schema   | public
target_table    | kafka_flex_0
timeslice       | 00:00:09.897
batch_start     | 2016-07-28 11:31:45.84898
batch_end       | 2016-07-28 11:31:46.253367
last_batch_duration | 000:00:00.377796
consecutive_error_count |
transaction_id  | 45035996275130109
frame_start     | 2016-07-28 11:31:45.751
frame_end       |
end_reason_message | Local: All brokers are down

```

stream_microbatch_source_map

This table maps microbatches to the their associated sources.

Column	Data Type	Description
microbatch	INTEGER	The identification number of the microbatch.
source	INTEGER	The identification number of the associated source.

Examples

This example shows typical rows from the `stream_microbatch` table.

```
SELECT * FROM stream_config.stream_microbatch_source_map;
microbatch | source
-----+-----
          1 |      4
          3 |      2
(2 rows)
```

stream_microbatches

This table contains metadata related to microbatches

Column	Data Type	Description
id	INT	The identification number of the microbatch.
microbatch	VARCHAR	The name of the microbatch.
target	INT	The identification number of the target associated with the microbatch.
load_spec	INT	The identification number of the load spec associated with the microbatch.
target_columns	VARCHAR	The table columns associated with the microbatch.
rejection_schema	VARCHAR	The schema that contains the rejection table.
rejection_table	VARCHAR	The table where Verticastores messages that are rejected by the database.
enabled	BOOLEAN	When TRUE, the microbatch is enabled for use.

Examples

This example shows a row from a typical `stream_microbatches` table.

```
SELECT * FROM stream_config.stream_microbatches;
-[ RECORD 1 ]-----
id           | 1
microbatch   | mbatch1
target       | 1
load_spec    | 750001
target_columns |
rejection_schema | public
rejection_table | rejected_messages
enabled      | t
```

stream_scheduler

This table contains metadata related to a single scheduler.

This table was renamed from `kafka_config.kafka_scheduler`.

Column	Data Type	Description
version	VARCHAR	The version of the scheduler.
frame_duration	INTERVAL	The length of time of the frame. The default is 00:00:10.
resource_pool	VARCHAR	The resource pool associated with this scheduler.
config_refresh	INTERVAL	The interval of time that the scheduler runs before applying any changes to its metadata, such as, changes made using the <code>--update</code> option. For more information, refer to <code>--config-refresh</code> in Scheduler Utility Options .
new_source_policy	VARCHAR	When during the frame that the source runs. Set this value with the <code>--new-source-policy</code> in Source Utility

Column	Data Type	Description
		<p>Options.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> FAIR: Takes the average length of time from the previous batches and schedules itself appropriately. START: Runs all new sources at the beginning of the frame. In this case, Vertica gives the minimal amount of time to run. END: Runs all new sources starting at the end of the frame. In this case, Vertica gives the maximum amount of time to run. <p>Default Value: FAIR</p>
eof_timeout_ms	INT	<p>The maximum amount of time the scheduler waits for data from the source before ending the batch.</p> <p>Default Value: 1000</p>
pushback_policy	VARCHAR	<p>How Vertica handles delays for microbatches that continually fail.</p> <p>Valid Values:</p> <ul style="list-style-type: none"> FLAT LINEAR EXPONENTIAL <p>Default Value: LINEAR</p>
pushback_max_count	INT	<p>The maximum number of times a</p>

Column	Data Type	Description
		microbatch can fail before Vertica terminates it.
auto_sync	BOOLEAN	When TRUE, the scheduler automatically synchronizes source information with host clusters. For more information, refer to Data Streaming Job Scheduler . Default Value: TRUE

Examples

This example shows a typical row in the stream_scheduler table.

```
SELECT * FROM stream_config.stream_scheduler;
-[ RECORD 1 ]-----+-----
version          | v8.0.0
frame_duration   | 00:00:10
resource_pool    | kafka_default_pool
config_refresh   | 00:05
new_source_policy | FAIR
eof_timeout_ms   | 10000
pushback_policy  | LINEAR
pushback_max_count | 5
auto_sync        | t
```

stream_scheduler_history

This table shows the history of launched scheduler instances.

This table was renamed from kafka_config.kafka_scheduler_history.

Column	Data Type	Description
elected_leader_time	TIMESTAMP	The time when this instance took began scheduling operations.
host	VARCHAR	The host name of the machine running the scheduler instance.

Column	Data Type	Description
launcher	VARCHAR	The name of the currently active scheduler instance. Default Value: NULL
scheduler_id	INT	The identification number of the scheduler.
version	VARCHAR	The version of the scheduler.

Examples

This example shows typical rows from the stream_scheduler_history table.

```

SELECT * FROM stream_config.stream_scheduler_history;
  elected_leader_time |      host      | launcher          | scheduler_id | version
-----|-----|-----|-----|-----
2016-07-26 13:19:42.692 | 10.20.100.62 |                  | 0 | v8.0.0
2016-07-26 13:54:37.715 | 10.20.100.62 |                  | 1 | v8.0.0
2016-07-26 13:56:06.785 | 10.20.100.62 |                  | 2 | v8.0.0
2016-07-26 13:56:56.033 | 10.20.100.62 | SchedulerInstance | 3 | v8.0.0
2016-07-26 15:51:20.513 | 10.20.100.62 | SchedulerInstance | 4 | v8.0.0
2016-07-26 15:51:35.111 | 10.20.100.62 | SchedulerInstance | 5 | v8.0.0
(6 rows)

```

stream_sources

This table contains metadata related to data streaming sources.

This table was formerly named kafka_config.kafka_scheduler.

Column	Data Type	Description
id	INTEGER	The identification number of the source
source	VARCHAR	The name of the source.
cluster	INTEGER	The identification number of the cluster associated with the source.
partitions	INTEGER	The number of partitions in the source.

Column	Data Type	Description
enabled	BOOLEAN	When TRUE, the source is enabled for use.

Examples

This example shows a typical row from the stream_sources table.

```
select * from stream_config.stream_sources;
-[ RECORD 1 ]-----
id          | 1
source      | SourceFeed1
cluster     | 1
partitions  | 1
enabled     | t
-[ RECORD 2 ]-----
id          | 250001
source      | SourceFeed2
cluster     | 1
partitions  | 1
enabled     | t
```

stream_targets

This table contains the metadata for all Vertica target tables.

The table was formerly named kafka_config.kafka_targets.

Column	Data Type	Description
id	INTEGER	The identification number of the target table
target_schema	VARCHAR	The name of the schema for the target table.
target_table	VARCHAR	The name of the target table.

Examples

This example shows typical rows from the stream_tables table.

```
=> SELECT * FROM stream_config.stream_targets;
-[ RECORD 1 ]-----+-----
id                | 1
target_schema     | public
target_table      | stream_flex1
-[ RECORD 2 ]-----+-----
id                | 2
target_schema     | public
target_table      | stream_flex2
```

Integrating with Apache Spark

Welcome to the Micro Focus Vertica Connector for Apache Spark Guide.

Vertica Connector for Apache Spark is a fast parallel connector that transfers data between the Vertica Analytics Platform and Apache Spark. This feature lets you use Spark to pre-process data for Vertica and to use Vertica data in your Spark application.

Apache Spark is an open-source, general-purpose cluster-computing framework. It evolved as a faster, multi-stage, in-memory alternative to the two stage, disk-based Map Reduce framework offered by Hadoop. The Spark framework is based on *Resilient Distributed Datasets* (RDDs), which are logical collections of data partitioned across machines. Spark is typically used in upstream workloads to process data before loading it in Vertica for interactive analytics. It can also be used downstream of Vertica, where data pre-processed by Vertica is then moved into Spark for further transformation.

Using the Micro Focus Vertica Connector for Apache Spark, you can:

- Move large volumes of data from Spark DataFrames to Vertica tables; the connector allows you to write Spark DataFrames to Vertica tables.
- Move data from Vertica to Spark RDDs or DataFrames for use with Python, R, Scala and Java. The connector efficiently pushes down column selection and predicate filtering to Vertica before loading the data.

Audience

This book is intended for anyone who wants to transfer data between a Vertica database and an Apache Spark cluster.

Prerequisites and Compatibility

This document assumes that you have installed and configured Vertica as described in [Installing Vertica](#) and the [Configuring the Database](#) section of the Administrator's Guide. You must also have installed your Apache Spark clusters.

To save data from Spark to Vertica, you must have an HDFS cluster for an intermediate staging location. Your Vertica database must be configured to read data from this HDFS cluster. See [Reading Directly from HDFS](#) in [Integrating with Apache Hadoop](#) for more information.

For details on installing and using Apache Spark and Apache Hadoop, see the [Apache Spark web site](#), the [Apache Hadoop website](#), or your Hadoop vendor's installation documentation.

For supported versions of Apache Spark and Apache Hadoop see the following sections in the [Supported Platforms guide](#):

- [Vertica Integration for Apache Spark](#)
- [Vertica Integrations for Hadoop](#)

The startup messages contain the version numbers of both Spark and Scala (shown in bold in the previous example for clarity).

Deploying the Vertica Connector for Apache Spark

Once you have downloaded the connector and JDBC library JAR files, you can deploy them to your Spark cluster in two ways:

- Include the connector and Vertica JDBC JAR files using the "--jars" option when invoking spark-submit or spark-shell.
- Deploy the connector to a Spark cluster so that all Spark applications have access across all nodes.

Copying the Connector for Use with Spark Submit or Spark Shell

1. Log on as the Spark user on any Spark machine.
2. Copy both the Vertica Spark Connector and Vertica JDBC Driver JAR files from the package to your local Spark directory.
3. Then, run the connector in any of the following ways (replace the version number in the jar names with your version number):
 - Run a Spark Application using spark-submit (<http://spark.apache.org/docs/latest/submitting-applications.html#launching-applications-with-spark-submit>). You must run the command on the node with the Vertica JAR files.

```
spark-submit --jars vertica-8.1.0_spark2.0_scala2.11.jar,vertica-jdbc-8.1.0-0.jar other options SparkApplication.jar
```

Note: Do not include a space before or after the comma in the --jars argument.

- Use the Interactive spark-shell (<http://spark.apache.org/docs/latest/programming-guide.html#using-the-shell>). You must run the command on the node with the Vertica JAR files:

```
spark-shell --jars vertica-8.1.0_spark2.0_scala2.11.jar,vertica-jdbc-8.1.0-0.jar other options
```

Note: The version numbers in the JAR file names will vary depending on your version of Vertica, Spark, and Scala.

Deploying the Connector to a Spark Cluster

You can optionally deploy the JAR files to a Spark cluster. This approach gives all applications (such as shell and submit) access; it does not require specifying them on the command line.

To deploy to the Spark cluster:

1. Copy the files to a common path on all Spark machines.
2. Add the path for the connector and JDBC driver to your `conf/spark-defaults.conf`, and restart the Spark Master. For example, modify the `spark.jars` line by adding the connector and JDBC JARS as follows (replace paths and version numbers with your values):

```
spark.jars /JAR_file_Path/vertica-8.1.0_spark2.0_scala2.11.jar,/JAR_file_Path/vertica-jdbc-8.1.0-0.jar
```

Saving an Apache Spark DataFrame to a Vertica Table

The Vertica Connector for Apache Spark copies data partitions distributed across multiple Spark worker-nodes into a temporary location in HDFS. Vertica then reads the data from HDFS. These data transfers use parallel reads and writes, letting the connector efficiently load large volumes of data from Spark to Vertica.

Save Options

When writing Spark DataFrames to Vertica, you specify a Vertica target table. You also set how the data is saved using the DataFrame [SaveMode](#). The valid values for the SaveMode are:

- **SaveMode.Overwrite** overwrites or create new table.

Note: The existing table is dropped whether the save succeeds or not, except when the connector is asked to load a DataFrame that contains zero rows.

- **SaveMode.Append** appends data to an existing table or creates the table if it does not exist.
- **SaveMode.ErrorIfExists** creates a new table if the table does not exist, otherwise returns an error.
- **SaveMode.Ignore** creates a new table if the table does not exist, otherwise it does not save the data and does not return an error.

The `save()` operation never results in a partial or duplicate save. The connector either saves the DataFrame in its entirety to the target table successfully or it aborts the save. In case of failures, to avoid leaving the target table in an inconsistent state, the connector:

1. Saves the data to HDFS as an intermediate staging location
2. Safely copies it into the actual target table in Vertica

Rejected Rows

For bulk loads, the connector API provides user control to specify a tolerance for rejected rows. You can specify the user tolerance as a parameter (see [Parameters](#) in [Saving Spark Data to Vertica Using the DefaultSource API](#)). If the number of rejected rows falls within the tolerance level, the save completes successfully. Otherwise, the connector aborts the save and reports an error.

Job Status

The connector creates a Vertica table, `S2V_JOB_STATUS_USER_$USERNAME$`, to report the status of each job. When the save to Vertica starts, the connector writes the unique `job_id` to the Spark log. After the Spark job is started, users can consult this table for the job status based on the `job_id`. The `S2V_JOB_STATUS_USER_$USERNAME$` has the following columns:

- `target_table_schema`
- `target_table_name`
- `save_mode`
- `job_name`
- `start_time`
- `all_done`
- `success`
- `percent_failed_rows`

The table indicates the start time, unique job name, date, percentage of rows that failed, and final outcome of the save job (success=T/F). The column `all_done` indicates the Spark job finished without errors and the column `success` indicates the data was saved to the Vertica table (rejected rows must fall within specified tolerance).

Saving Spark Data to Vertica Using the DefaultSource API

The Vertica Connector for Apache Spark provides the `com.vertica.spark.datasource.DefaultSource` API to simplify writing data from a Spark DataFrame to a Vertica table using the Spark `df.write.format()` method. The DefaultSource API provides generic key-value options for configuring the database connection and tuning parameters as well as other options.

Requirements

The following requirements apply to using the connector to save Spark data to Vertica.

- When you append to an existing Vertica table (using **SaveMode.Append**), your DataFrame must have the same column types, column order, and number of columns as the Vertica table.
- The specified Vertica username must have CREATE privileges on the Vertica schema. The connector defaults to "public" schema, but any schema name can be provided as an option by using the "dbschema" optional argument.
- A Spark SaveMode must to be specified (Append, Overwrite, Ignore, ErrorIfExists). See [Saving an Apache Spark DataFrame to a Vertica Table](https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/sql/SaveMode.html) and <https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/sql/SaveMode.html>.

Limitations

The following limitations apply to using the connector to save Spark data to Vertica.

- **Supported Data Types**—The Spark DataFrame cannot have complex types such as Maps, Structs, and Arrays. Vertica currently supports all other Spark data types. The connector cannot save a DataFrame that contains any of these types.
- **Target Table Limitations**—The Vertica target table specified cannot be a view or a temporary table.

- **Null Values in Spark and Vertica**—Spark's long type is converted to Vertica's INT type when data moves from Spark to Vertica. Vertica uses the value -2^{63} (-9223372036854775808) to represent a NULL value. Spark does not treat this value in any special way. If you save -2^{63} from Spark to Vertica then it is stored as NULL in Vertica.
- **SQL Strings**—Length of Strings. Currently the connector converts all Spark SQL Strings to VARCHAR(strlen) for Vertica. The 'strlen' parameter is a user option that defaults to 1024. If strlen is set to a value greater than 65000, Spark SQL Strings are instead converted to LONG VARCHAR(strlen). Values in the range 1 to 32,000,000 are valid.

When using SaveMode.Append, the existing Vertica table should have the corresponding column types for Spark SQL Strings declared as either VARCHAR or LONG VARCHAR.

Spark SQL Strings longer than strlen are always truncated when saving to Vertica.

- **Float Values**—Float Values Represented Differently. Because float values are approximations, the value of a float can be different when it is moved from Spark to Vertica. If you require more precision, then use a more precise data type in Spark, such as double or decimal.
- **Gregorian/Julian Calendar Issues**—Inconsistent dates before the year 1583. The ORC file writer used to stage the data to in HDFS converts dates before 1583 from the Gregorian calendar to the Julian calendar. However Vertica does not perform this conversion. If your data in Spark contains dates before 1583, then the values in Spark and the corresponding values in Vertica can differ by up to 10 days. This difference applies to both DATE and TIMESTAMP values.
- **Wrong results for Decimal Type.** If your version of Vertica is prior to 7.2 SP3, then decimal numbers are incorrectly loaded from Spark to Vertica. Upgrade to a newer version of Vertica to resolve this issue.
- **Different Jobs Writing to the Same Table in Overwrite Mode cause error:**—ERROR: java.sql.SQLException: [Vertica][VJDBC](3007) ERROR: DDL statement interfered with this statement. If you have different jobs writing to the same table and are using Overwrite mode then errors can occur. For example, Job 1 starts and begins writing to the Vertica table. Then Job 2 begins and is set to write to the same table. Because the mode is Overwrite, it drops the table to which Job 1 is writing, causing an error. Do not use different jobs to write to the same table when using overwrite mode.
- **Files not cleaned up if process interrupted.**—The files written on by Spark to HDFS are normally cleaned up when the job completes. However, if something interrupts the process

(such as a network failure), then files may be left behind and you should manually cleanup the HDFS directory. The HDFS directory is the one provided in the `hdfs_url` path, and the specific subdirectory is `S2V_jobxyz` as reported by the connector.

- **Error if the DataFrame has zero rows**—The connector returns an error message asking you to check whether the DataFrame is empty. This error occurs before the connector begins loading data into Vertica. The data in the targeted Vertica table is not altered, even if mode was set to `SaveMode.Overwrite` (which would normally delete the table).

Parameters

To save your Spark DataFrame to Vertica, you must specify the following required parameters as options to `sqlContext.createDataFrame(...).format("com.vertica.spark.datasource.DefaultSource").options(...)`:

Parameter	Description
<code>table</code>	The name of the target Vertica table to save your Spark DataFrame.
<code>db</code>	The name of the Vertica Database
<code>user</code>	The name of the Vertica user. This user must have CREATE and INSERT privileges in the Vertica schema. The schema defaults to “public”, but may be changed using the “dbschema” optional tuning parameter.
<code>password</code>	The password for the Vertica user.
<code>host</code>	The hostname of a Vertica node. This value is used to make the initial connection to Vertica and look up all the other Vertica node IP addresses. You can provide a specific IP address or a resolvable name such as <code>myhostname.com</code> .
<code>hdfs_url</code>	<p>The fully-qualified path to a directory in HDFS that will be used as a data staging area. For example, <code>hdfs://myhost:8020/data/test</code>.</p> <p>The connector first saves the DataFrame in its entirety to this location before loading into Vertica. The data is saved in ORC format, and the files are saved in a directory specific to each job. This directory is then deleted when the job completes.</p> <p>Note that you need additional configuration changes for to use HDFS. You must first configure all nodes to use HDFS. See Configuring the hdfs Scheme.</p>

Parameter	Description
	<p>You can optionally use the webhdfs protocol instead. See HDFS and WebHDFS.</p> <p>Note: You must provide an HDFS URL even if you intend to use webhdfs. You can use a dummy URL.</p>

In addition to the required parameters, you can optionally specify the following parameters as options to `sqlContext.createDataFrame(...).format("com.vertica.spark.datasource.DefaultSource").options(...)`.

Parameter	Description
<code>dbschema</code>	The schema space for the Vertica table. Default Value: <code>public</code>
<code>port</code>	The Vertica Port. Default value: 5433
<code>failed_rows_percent_tolerance</code>	The tolerance level for failed rows, as a percentage. For example, to specify that the job fail if greater than 10% of the rows are rejected, specify this value as 0.10 for 10% tolerance. Default Value: 0.00
<code>strlen</code>	The string length. Use this option to increase (or decrease) the default length when saving Spark StringType to Vertica VARCHAR type. Default Value: 1024
<code>web_hdfs_url</code>	The fully-qualified path to a directory in HDFS that will be used by Vertica to retrieve the data. For example, <code>webhdfs://myserver:50070/data/test</code> . You must use this option (in addition to "hdfs_url") if the Vertica nodes are not configured for HDFS access. See HDFS and WebHDFS . below.

HDFS and WebHDFS

The connector transfers data from Spark to an HDFS directory before moving it to Vertica. Vertica can access data in the HDFS directory either directly using the `hdfs` scheme, or through the web-based `webhdfs` scheme. For greater performance and reliability, Vertica should use direct HDFS access whenever possible.

You must configure your Vertica cluster before it can directly access data stored in HDFS. See [Configuring the hdfs Scheme](#) in the [Integrating with Apache Hadoop](#).

By default, the connector uses the `hdfs` protocol when possible. It falls back to the `webhdfs` protocol in several cases:

- If you do not have your nodes configured for direct HDFS access.
- If the data stored in the HDFS directory is encrypted. The library that Vertica uses to read directly from HDFS cannot decrypt this data. The `webhdfs` scheme transparently decrypts the data for Vertica.

For Vertica to fall back to `webhdfs`, you must verify that `webhdfs.enabled` is set to `true` on your HDFS cluster.

Quick Start with Code Example

You can use the `spark-shell` and some brief Scala code to verify that the connector can write data from Spark to Vertica.

1. Start the `spark-shell` and include both the Vertica JDBC Driver and the Vertica Spark Connector JAR files in the `jars` argument. Then, specify any other Spark options you normally use:

```
spark-shell --jars vertica-8.1.0_spark2.0_scala2.11.jar,vertica-jdbc-8.1.0-0.jar other options
```

2. The following code creates a `DataFrame` in Spark and saves it to a new Vertica table using the connector:
 - a. Modify the `host`, `db`, `user`, `password`, and `table` settings in the following example to match your Vertica instance.
 - b. Paste the following code into your Spark shell.

```
// S2V_basic.scala
import org.apache.spark.sql.types._
import org.apache.spark.sql.{DataFrame, Row, SQLContext, SaveMode}
import com.vertica.spark._

// Create a sample DataFrame and save it to Vertica
val rows = sc.parallelize(Array(
  Row(1,"hello", true),
  Row(2,"goodbye", false)
))

val schema = StructType(Array(
  StructField("id",IntegerType, false),
  StructField("message",StringType,true),
  StructField("still_here",BooleanType,true)
```

```
))

// Note: Spark's API changed between version 1.6 and 2.0. In version 1.6
// you use sqlContext to create a DataFrame. In 2.0, you use the spark object.
// val df = sqlContext.createDataFrame(rows, schema) // Spark 1.6
val df = spark.createDataFrame(rows, schema) // Spark 2.0

// View the sample data and schema
df.show
df.schema

// Setup the user options, defaults are shown where applicable for optional values.
// Replace the values in italics with the settings for your Vertica instance.
val opts: Map[String, String] = Map(
  "table" -> "VerticaTableName",
  "db" -> "VerticaDatabaseName",
  "user" -> "VerticaDatabaseUser",
  "password" -> "VerticaDatabasePassword",
  "host" -> "VerticaHostName",
  "hdfs_url" -> "hdfs://HDFSNameNode:9000/user/hduser/someDirectory",
  "web_hdfs_url" -> "webhdfs://HDFSNameNode:50070/user/hduser/someDirectory"
  // "failed_rows_percent_tolerance" -> "0.00" // OPTIONAL (default val shown)
  // "dbschema" -> "public" // OPTIONAL (default val shown)
  // "port" -> "5433" // OPTIONAL (default val shown)
  // "strlen" -> "1024" // OPTIONAL (default val shown)
)

// SaveMode can be either Overwrite, Append, ErrorIfExists, Ignore
val mode = SaveMode.Overwrite

// save the DataFrame via Spark's Datasource API
df.write.format("com.vertica.spark.datasource.DefaultSource").options(opts).mode(mode).save
()
```

Loading Vertica Data into a Spark DataFrame or RDD

The Vertica Connector for Apache Spark includes APIs to simplify loading Vertica table data efficiently with an optimized parallel data-reader:

- `com.vertica.spark.datasource.DefaultSource` — The data source API, which is used for writing to Vertica and is also optimized for loading data into a DataFrame.
- RDD API `com.vertica.spark.rdd.VerticaRDD` — This API simplifies creating an RDD object based on a Vertica table or view.

Typically, Vertica tables are segmented across multiple nodes. Using the Spark connector, you invoke a parallel data reader to efficiently read data from Vertica by minimizing data movement between Vertica nodes. The DataFrame reader supports pushing column and row filtering to Vertica to avoid transferring large volumes of Vertica data into the Spark in-memory data structures.

Important: For the best possible performance, segment your Vertica table by hash on one or more attributes that return integer values. Hewlett Packard Enterprise does not recommend segmenting by a custom expression, because doing so can result in lower performance than segmenting by hash. See [Hash Segmentation Clause](#) for more details. An example of creating a table segmented by hash:

```
create table example(a integer, b integer) segmented by hash(a) all nodes;
```

Loading Vertica Table Segments into the Spark DataFrames and RDD Partitions

All Spark RDDs and DataFrames require you to define the number of partitions. You can define an arbitrary number of partitions. The Vertica Spark connector library automatically generates the hash-intervals for each partition and intervals. When you segment the table by a proper hash expression on one or more of its columns, these intervals minimize cross-node data shuffling (*w*) inside Vertica and data skew.

No cross-node data shuffling occurs inside Vertica when both of the following conditions exist:

- The Vertica cluster has N nodes.
- The number of the Spark partitions is $P * N$ (P is an integer greater than 0).

Therefore, you can use the full network throughput for data transfer, achieving the best performance. The following figure shows 8 Spark partitions defined over 4 Vertica segmentations.



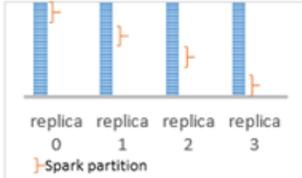
The next figure shows another example where the number of Spark partitions is smaller (two) than the number of Vertica segmentations (four).



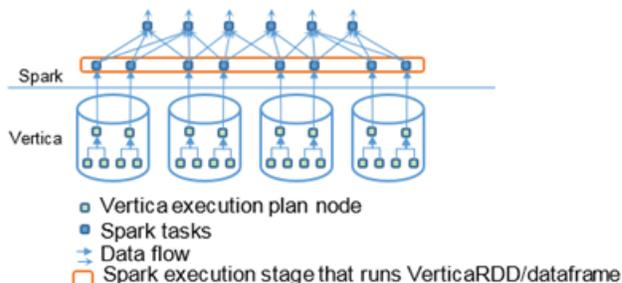
According to Vertica benchmark tests, a setting of $P=4$ achieves the best loading performance. For example, when Vertica has 4 nodes, the number of Spark partitions recommended is 16 ($P = 4, Nodes = 4, P * N = 16$ partitions).

During Spark job execution, each Spark partition executes inside a task. Each task launches a JDBC connection to a Vertica node. This connection contacts the Vertica node that stores the segment containing the data of the Spark partition. The Spark task then issues a query to the Vertica node and fetches the query result into a VerticaRDD/DataFrame.

For an unsegmented Vertica table, the connector replicates the table onto multiple Vertica nodes. The Spark partition is defined as a range on the sorted columns of the table. Spark tasks send the queries to different replicas for load-balancing. The next figure shows Spark partitions defined over an unsegmented table.



The following figure illustrates the runtime behavior of the Vertica to Spark connector. Spark tasks containing VerticaRDD/DataFrame partitions fetch data from Vertica through JDBC connections. Those queries that are used to fetch data are based on the computed ranges of hash values.



Spark Numeric Value Limitation

Both Vertica and Spark support variable-precision NUMERIC values. Vertica's **NUMERIC** values support more digits of precision (up to 1024) than Spark does (38 digits). If you attempt to copy a Vertica table to Spark that has a NUMERIC column with more than 38 digits of precision, the `VerticaDataSourceRDD` class throws an error similar to the following:

```
java.lang.IllegalArgumentException: requirement failed: Decimal precision 41 exceeds max precision 38
    at scala.Predef$.require(Predef.scala:224)
    at org.apache.spark.sql.types.Decimal.set(Decimal.scala:113)
    at org.apache.spark.sql.types.Decimal$.apply(Decimal.scala:426)
    at com.vertica.spark.datasource.VerticaDataSourceRDD$.anon$.getNext(VerticaRDD.scala:382)
    . . .
```

Loading Vertica Data into a Spark DataFrame Using the Vertica Data Source API

The Vertica Connector for Apache Spark data source API supports both parallel write and read operations. The following code sample illustrates how you can create an in-memory DataFrame by invoking `SQLContext.read` function, using Vertica's `com.vertica.spark.datasource.DefaultSource` formatter.

Parameters

To connect to Vertica, you must specify the following parameters for the options in `sqlContext.read.format`

```
("com.vertica.spark.datasource.DefaultSource").options(...).
```

Parameter	Description
table	The name of the target Vertica table or view to save your Spark DataFrame. Note that the Vertica table must be segmented by hash or by an expression that returns non-negative integer values. Also, if the Vertica table is an external table, the underlying file(s) on which the external table is based must be accessible on all Vertica nodes.
db	The name of the Vertica Database
user	The name of the Vertica user. This user must have CREATE and INSERT privileges in the Vertica schema. The schema defaults to “public”, but may be changed using the dbschema optional parameter.
password	The password for the Vertica user.
host	The hostname of a Vertica node. This value is used to make the initial connection to Vertica and look up all the other Vertica node IPs. You can provide a specific IP address or resolvable name such as myhostname.com.
numPartitions	Optional. The number of Spark partitions for the load from the Vertica job, each partition creates a JDBC connection. Default Value: 16.
dbschema	Optional. The schema space for the Vertica table. Default value: public.
port	Optional. The Vertica Port. Default value: 5433

Example: Load Data into a DataFrame

The following example demonstrates reading the content of a Vertica table named test into a DataFrame.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.sql._
import org.apache.spark.SparkConf

// Note: the following is deprecated in Spark 2.0. It will warn you to use
// SparkSession instead.
val sqlContext = new SQLContext(sc)

val table = "test"
val db = "myDB"
```

```
val user = "myUser"
val password = "myPassword"
val host = "myVerticaHost"
val part = "12";

val opt = Map("host" -> host, "table" -> table, "db" -> db, "numPartitions" -> part, "user" -> user,
"password" -> password)

val df = sqlContext.read.format("com.vertica.spark.datasource.DefaultSource").options(opt).load()
```

Column Selection and Filter Push Down

Column selections and row filters applied on the DataFrame are pushed down into Vertica. Refer to the Spark SQL API for more details about applying column selections and row filters. The following example illustrates how you can load a column containing filtered rows as a DataFrame:

```
val c = df.select("a").filter("a > 5").count
```

Note: When filtering it is important to specify the correct type. For example, single-quoted values are treated as strings, so if you wrote the above filter code as `filter("a > '5'")` then the filter is not pushed down, because the target column is an int and the value '5' when single-quoted is treated as a string.

Additionally, when using other types such as dates, cast the value as a date type, such as: `filter("c1 >= cast('2010-1-2' as date)` rather than `filter("c1 >= '2010-1-2')`. The latter example does not push down.

Additional examples of valid filters:

- `df.filter("id<=3").groupBy("id").sum("id").show`
- `df.filter($"c".like("str%")).show` // has to be varchar type to be pushed down
- `df.filter($"c".rlike("str")).show` // has to be varchar type to be pushed down
- `df.filter($"id".isin(3,5)).show`

Considerations When Using the Spark DataFrame Filter Method with Vertica

Be aware of the following considerations when using the DataFrame's filter method when loading data from Vertica:

- Vertica supports the case insensitive terms “inf” and “infinity” to refer to INFINITY in SQL queries. However, when filtering using Spark’s filter method you must instead use the term (case sensitive) “Infinity”.
- When filtering on Boolean values, do not use "is true/false" or "is not true/false" instead, cast the value as a Boolean, or use the equals operator, for example:
 - `df.filter("c = 1").show`
 - `df.filter("c = True").show`
 - `df.filter("c = False").show`

Example: Creating a DataFrame Using the Vertica Data Source

Use the following example to learn how to create a Spark DataFrame from a Vertica table:

1. Create a sample table in Vertica:

```
=> CREATE TABLE test (a int, b int, c int, d varchar);
=> INSERT INTO test VALUES (1, 3, 5, 'odds');
=> INSERT INTO test VALUES (10, 14, 8, 'evens');
=> INSERT INTO test VALUES (11, 13, 19, 'odds');
=> COMMIT;
```

2. Modify the database connection details (host, db, table, user, and password) in the code below with your Vertica connection details.

```
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

val conf = new SparkConf().setAppName("vertica-spark-connector-testing").setMaster("local[1]")
val sc = new SparkContext(conf)
```

```
val sqlContext = new SQLContext(sc)

val host = "VerticaHost"
val db = "VerticaDB"
val table = "VerticaTable"
val user = "VerticaUser"
val password = "VerticaPassword"
val part = "12";

val opt = Map("host" -> host, "table" -> table, "db" -> db, "numPartitions" -> part, "user" ->
user, "password" -> password)

val df = sqlContext.read.format ("com.vertica.spark.datasource.DefaultSource").options(opt).load
()

val c = df.select("a").filter("a > 5").count
println(c)
sc.stop();
```

3. Run the example code in a Spark shell. Start the shell, then paste in your modified code.

Loading Vertica Data into a Spark RDD Using the Vertica RDD API

Use the Vertica RDD API to load Vertica table data into a Spark RDD. You create a `VerticaRDD` object either by initiating it or by calling one of the multiple `RDD.create` methods available. The following example uses `VerticaRDD.create` to create an RDD from Vertica data. In this example, the parameters of `create()` are:

Parameter	Description
hostname	A Vertica node host name
port	Vertica port number. The default value is 5433.
prop	A property object that include user name and password
table	Vertica table name, including the schema name, such as, "schema_name.table_name".
dbname	Vertica database name
col	The column names of the Vertica table that will be loaded into Spark. An empty col array means all columns.

Parameter	Description
numPartitions	The number of Spark partitions for the resulting VerticaRDD
mapRow	<p>A function used to convert one row of JDBC results into the element data type T of VerticaRDD. The generated VerticaRDD has the type VerticaRDD[T].</p> <p>For example:</p> <pre>val extractValues = (r: ResultSet) => { (r.getInt(1), r.getInt(2)) }</pre> <p>This function converts a tuple that contains two integers—for example, (101, 102), into an array of objects. Refer to the Spark JDBC RDD for more details about this parameter.</p> <p>Note: This function must be serializable by Spark.</p>

RDD Create Methods

The `com.vertica.spark.rdd.VerticaRDD` API has three different create methods that you can use to create a VerticaRDD object:

- `create(sc, connect, table, columns, numPartitions, mapRow)`
- `create(sc, connect, table, columns, numPartition, ipMap, mapRow)`
- `create(sc, host, port: Int = 5433, db, properties, table, columns, numPartitions, mapRow)`

The parameters for the Create methods:

Parameter	Description
sc	Spark Context object.
connect	A connection object that contains a function that returns an open JDBC Connection.
table	A string value for the database table name
columns	An <code>Array[String]</code> that contains the columns of the database table

Parameter	Description
	that will be loaded. If an empty array, this RDD loads all columns.
numPartitions	An integer value specifying the number of partitions for the VerticaRDD.
ipMap	<p>A Map [String, String] containing the optional map from private IP addresses to public IP addresses for all Vertica nodes. This map is used only when Vertica is installed on private IP addresses but is listening on both private and public IP addresses.</p> <p>You can also update each Vertica node's EXPORT_ADDRESS instead of providing the ipMap parameter if Vertica is running on both private and public IP addresses.</p> <p>When a database is installed, the export_address and node_address in the system NODES table are set to the same value. If you installed Vertica on a private address, you must set the export_address to a public address for each node.</p> <p>See Identify the Database or Nodes Used for Import/Export in the Administrator's Guide.</p>
mapRow	A function from a ResultSet to a single row of the upi result types you want. This function should call only getInt, getString, etc; the RDD takes care of calling next. The default maps a ResultSet to an array of Object.
host	A string containing the host name (or IP address) of a Vertica node, that can be used to connect to Vertica.
port	An integer value specifying the number for Vertica connection. Default Value: 5433
properties	Properties object for JDBC connection properties, such as user, password.

Example: Loading Vertica Table Data into a VerticaRDD

This section contains complete example code used to create a VerticaRDD from a Vertica table, test whose definition is:

```
create table test (a int, b int);
```

This program creates `VerticaRDD[(Int, Int)]` and calls `count` that returns the number of rows of table, `test`.

```
// V2S_rdd.scala
import com.vertica.spark.rdd.VerticaRDD
import java.util.Properties
import java.sql.ResultSet
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

val extractValues = (r: ResultSet) => {
  (r.getInt(1), r.getInt(2))
}

val conf = new SparkConf().setAppName("vertica-spark-connector-testing").setMaster("local[1]")
//val sc = new SparkContext(conf) // uncomment if not used in spark shell

val host = "VerticaHost"
val port = 5433
val db = "VerticaDB"
val prop = new Properties
prop.put("user", "VerticaUserName")
prop.put("password", "VerticaPassword")
val table = "test"
val cols = Array[String]()
val part = 12;
val data = VerticaRDD.create(sc, host, port, db, prop, table, cols, numPartitions = part, mapRow =
extractValues)
val c = data.count
println("count:" + c)
```


Vertica Pulse

Welcome to the Vertica Pulse Guide. This book describes how to use Vertica Pulse.

Audience

This book is intended for anyone who wants to use the sentiment analysis features in Vertica.

Prerequisites

This document assumes that you have installed and configured Vertica as described in [Installing Vertica](#) and the [Configuring the Database](#) section of the [Administrator's Guide](#).

Additionally, you must install a compatible Java JRE or JDK on all nodes in the cluster and configure Vertica to use Java for Java UDxs.

Please see the [Vertica product documentation](#) to learn more.

Pulse Virtual Machine Quick Start

These Quick Start instructions detail the minimal steps for installing and using Pulse with the Vertica Virtual Machine Image. Consult the complete documentation for detailed steps on installing Pulse on your own platform.

Downloading and Installing Pulse

1. Go to <http://my.vertica.com/> and sign in. Then, click the **Download** tab.
2. Scroll down to the section "Download Vertica 7.1 Virtual Machines" and click the download link for your VM environment. These instructions assume you are installing the VMDK version - *VMWare Server 2.0 and Workstation 7.0*.
3. After the download completes, unzip the file.
4. Double-click the .vmx file in `vmsrvr_64/Vertica 7.1.x x64` for VMware. The VM starts in your VMWare application.
5. You are automatically logged in as `dbadmin`. However, the password for the user (and root) is 'password'.
6. In the VM, select **Applications > Accessories > Terminal** to open a terminal.
7. In the terminal, type `admintools` to start the administration tools.
8. You are prompted for a license when `admintools` starts for the first time. To use the community edition license, simply click OK. You are then prompted to accept the EULA. Accept the EULA then exit `admintools`.
9. As `dbadmin`, using `vsq1` on any node in the cluster, set the `JavaBinaryForUDx` Configuration Parameter (use `which java` to determine your java location):

```
vsq1 -t -c "ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';"
```

10. Copy the Vertica Pulse install package to the VM then, as root, install the Pulse Package:

```
rpm -Uvh /path/to/vertica-pulse.x86_64.xxx.rpm
```

Note: Only install Vertica Pulse on a single node. All Pulse functions are available on all nodes. However, the installation SQL scripts and user-dictionary loading script are only

available on the node on which you install the Pulse package.

11. As dbadmin, run the Pulse install script on the node on which you installed the Pulse Package:

```
vsq1 -f /opt/vertica/packages/pulse/ddl/install.sql
```

Using Pulse

1. Run a sentiment function:

```
select sentimentanalysis('Cookies are sweet.') OVER(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | cookies  |                1
(1 row)
```

Note: By default, VerticaPulse analyzes English text, however, you can also specify the language of the text being analyzed as an attribute of the sentimentanalysis() function. For example:

```
select sentimentanalysis('Cookies are sweet.', 'english') OVER(PARTITION BEST);
```

English and Spanish are the supported languages.

About the Vertica Pulse Package

Vertica Pulse provides a suite of functions that allow you to analyze and extract the sentiment from English and Spanish language text directly from your Vertica database.

Vertica Pulse features include:

- **Attribute based sentiment scoring** - Pulse scores the sentiment of attributes in a sentence. Attributes are generally nouns and are automatically discovered by Pulse. Pulse typically scores sentiment from a range of -1 (negative sentiment) to +1 (positive sentiment). A sentiment of 0 is considered neutral. Scoring individual attributes in a sentence instead of scoring the sentence as a whole provides a more granular analysis for the text. For example, consider the sentence "The quick brown fox jumped over the lazy dog." It would be difficult to score the sentiment on the sentence as a whole, but if you score on the attributes of fox and dog, you could say the sentiment on the fox was positive (the fox is quick), and the sentiment on the dog is negative (the dog is lazy).
- **Tuning to your domain** - Pulse provides functionality to recognize attributes that are specific to your domain. For example, you can add the name of your product or company to a 'white_list' so that it is discovered by Pulse.
- **Tuning of how sentiment is scored** - Pulse includes user-dictionaries of words that are used to help score sentiment. You can alter these user-dictionaries to fine tune the way your text is analyzed.
- **Filtering of attributes you are not interested in** - Pulse supports a special 'stop words' user-dictionary to indicate attributes that should not be analyzed. Alternately, you can choose to score sentiment only on attributes defined in your white_list.
- **Synonym mappings** - Pulse provides customizable mappings so that you can map synonyms to a base word, and then normalize the analysis for the synonyms to the base word. For example, you can map *Hewlett Packard* to *HP*.

Vertica Pulse requires that Java and the Vertica Java Support Package are installed on all nodes in the Vertica cluster.

Depending on the version of Pulse, it may support only one language (English or Spanish) or multiple languages (English and Spanish). For multilingual versions, Pulse can analyze each text row (for example a tweet) in the language of the text specified as argument, the language specified by the user as parameter or the default language. See [Multilingual Pulse](#) for details.

Installing or Upgrading Vertica Pulse

The Vertica Pulse Package requires that Java be installed prior to installing Vertica Pulse.

Vertica Pulse Package Version Requirements

Your server must be running version 7.1.x or later to run Pulse. Pulse must be installed on a Vertica node.

You can download the Vertica server package and from the [Vertica Marketplace](#).

Installation Overview

1. Verify that your Vertica server version matches your Vertica Pulse version.
2. [Install Java](#) on all Hosts and set the JavaBinaryForUDx Vertica configuration parameter to your Java binary location. For example, using vsql: ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java'
3. [Install the Vertica Package](#) on a single node in the cluster. The process is the same for installation or upgrade. You need only install it on a single node, but note that the SQL scripts used to install and uninstall the Pulse functions and the SQL script that creates pulse schema and the user-dictionaries tables are only available from the node on which you installed the Pulse package. The Pulse functions, once installed, are available on all nodes regardless if the package is installed on the node to which you are connecting.
4. [Modify the jvm resource pool](#) so that Pulse performs optimally on your system hardware.

Installing Java on Vertica Hosts

You must install a Java Virtual Machine (JVM) on every host in your Vertica cluster in order to run Pulse. Pulse requires a 64-bit Java Standard Edition 6, 7, or 8 (Java version 1.6, 1.7, 1.8) runtime. Both the Oracle JDK and openjdk are supported. You can choose to install either the Java Runtime Environment (JRE) or Java Development Kit (JDK), since the JDK also includes the JRE. See the [Java Standard Edition \(SE\) Download Page](#) to download an Oracle installation package for your Linux platform, or use your platforms packaging tool (such as yum or apt-get) to get a Java 1.6, 1.7, or 1.8 compatible version of open-jdk.

Once you have installed a JVM on each host, ensure that the `java` command is in the search path and calls the correct JVM by running the command:

```
java -version
```

This command should print something similar to:

```
java version "1.6.0_37"  
Java(TM) SE Runtime Environment (build 1.6.0_37-b06)  
Java HotSpot(TM) 64-Bit Server VM (build 20.12-b01, mixed mode)
```

Setting the JavaBinaryForUDx Configuration Parameter

The `JavaBinaryForUDx` configuration parameter tells Vertica where to look for the JRE to execute Java UDFs. After you have installed the JRE on all of the nodes in your cluster, you need to set this parameter to the absolute path of the Java executable. You can use the symbolic link that some Java installers create (for example `/usr/bin/java`). If the Java executable is in your shell search path, you can get the path of the Java executable by running the following command from the Linux command line shell:

```
$ which java  
/usr/bin/java
```

If the `java` command is not in the shell search path, use the path to the Java executable in the directory where you installed the JRE. For example, if you installed the JRE in `/usr/java/default` (which is where the installation package supplied by Oracle installs the Java 1.6 JRE), the Java executable is `/usr/java/default/bin/java`.

You set the configuration parameter by executing the following statement as a database superuser:

```
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
```

See [ALTER DATABASE](#) for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the [CONFIGURATION_PARAMETERS](#) system table:

```
=> \x
Expanded display is on.
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name =
'JavaBinaryForUDx';
-[ RECORD 1 ]-----+-----
node_name          | ALL
parameter_name    | JavaBinaryForUDx
current_value      | /usr/bin/java
default_value      |
change_under_support_guidance | f
change_requires_restart | f
description        | Path to the java binary for executing
                    | UDX written in Java
```

Once you have set the configuration parameter, Vertica will be able to find the Java executable on each node in your cluster in order to execute Java UDFs.

Note: Since the location of the Java executable is set by a single configuration parameter for the entire cluster, you must ensure that the path to the Java executable is the same across all of the nodes in the cluster.

Installing or Upgrading the Vertica Pulse Package on Your Host

After you install a JVM on all of the nodes in your cluster, you must install the Pulse Package on a single node. If upgrading, install the new package on the same host on which you previously installed the package. Pulse installation or upgrade is a two-step process:

1. Install/Update the RPM or DEB package for Pulse.
2. Run included sql scripts to install or update the Pulse functions and create the user dictionaries.

The Pulse install process installs the functions and schema required for sentiment analysis. You need only install it on a single node. However, be aware that the following SQL scripts are only available from the node on which you installed the Pulse package:

- SQL scripts used to install and uninstall the Pulse functions
- SQL script that populates and loads the dictionaries

You can access Pulse functions on all nodes, regardless if the package is installed on the node to which you are connecting.

Install or Upgrade the Pulse Package

When you upgrade or reinstall Pulse, it automatically uses port 5433 for vsql. If you are using a different port, configure it using the command `export VSQL_PORT=<port_number>`.

1. Copy the RPM or DEB package to the node where you want to install or upgrade Pulse. If you are upgrading Pulse then copy the new package to the same node where you previously installed the Pulse package. The version of Vertica Pulse must match the version of the Vertica server. For example, if your Vertica server is version 7.1.0, then the VerticaPulse version must also be 7.1.0.

If you are upgrading Pulse, you can find the currently-installed version number of Pulse with the command:

```
select lib_version, lib_sdk_version from user_libraries where lib_name = 'SentimentLib';
```

2. Log into the host and install the package.

- For Red Hat, use:

```
sudo rpm -Uvh /path-to-package/vertica-pulse.x86_64.xxx.rpm
```

- For Debian, use:

```
sudo dpkg -i /path-to-package/vertica-pulse.x86_64.xxx.deb
```

The Pulse Package is installed to `/opt/vertica/packages/pulse`.

After you install the package, you must run the appropriate SQL scripts to install or upgrade the Pulse functions and install the dictionary tables. Vertica automatically reloads any labeled user-defined dictionaries.

Running the Pulse Install Script

Run the install script to install or upgrade the Pulse functions and schema for the dictionaries and mappings required for sentiment analysis. You must run the install script once on the node on which you installed the package. After you run the install script, then all nodes can use the Pulse functions.

Important! Before running the install script, you must set the `JavaBinaryforUDx` configuration parameter or the install script fails to install the Pulse functions. See [Installing Java on Vertica Hosts](#).

To run the install script:

1. As the `dbadmin` user, on the node on which you installed the Pulse RPM/DEB, run the `install.sh` script:

```
bash /opt/vertica/packages/pulse/install.sh
```

Note: You must run the install script for installs or upgrades.

2. The script installs/upgrades the Pulse functions:

```
CREATE LIBRARY
CREATE TRANSFORM FUNCTION
etc...
```

3. If this is a fresh installation, then [Modify the jvm Resource Pool](#) to match your system hardware.

Tuning the `jvm` Resource Pool for Vertica Pulse

Note: You must modify the `jvm` resource pool to match the capabilities of your hardware so that Vertica Pulse has adequate resources to perform queries. If a cluster does not have sufficient resources to run a Vertica Pulse query, then such a query can fail with an Out Of Memory (OOM) exception.

Vertica Pulse runs as a Java UDX (User Defined eXtension) and uses the `jvm` resource pool to define the resources available to run Vertica Pulse queries.

Vertica starts a Java Virtual Machine (JVM) when you perform a Vertica Pulse query. The session from which you issue the query reserves resources for the JVM (across all nodes in the cluster) and it releases the resources when the session ends. You can also explicitly close the JVM attached to the session by using the command `SELECT release_jvm_memory();`

The most critical resource pool settings that affect Vertica Pulse are `MAXMEMORYSIZE` and `PLANNEDCONCURRENCY`.

- `MAXMEMORYSIZE` defines the amount of RAM that a JVM can use. By default `MAXMEMORYSIZE` is set to either 10% of system memory or 2GB, whichever is smaller.
- `PLANNEDCONCURRENCY` defines how many JVMs are allowed to run across the cluster and how many Pulse sessions you are able to run cluster-wide. By default, `PLANNEDCONCURRENCY` is set to `AUTO`, which is the lower of either the number of cores on the node, or memory / 2GB, but it is never automatically set to less than "4".

The amount of memory that each JVM is allocated is determined by `MAXMEMORYSIZE / PLANNEDCONCURRENCY`. For example, suppose `MAXMEMORYSIZE` is set to 8G and `PLANNEDCONCURRENCY` is set to 2. In this case, only a maximum of 2 sessions can run Vertica Pulse queries and the session JVMs have a maximum memory size of 4GB.

Tip: The basic thing to remember is that `PLANNEDCONCURRENCY` controls the number of sessions across the entire cluster that can run the `sentimentAnalysis()` function. If set to 1, then only a single session can run Pulse functions. No other sessions are able to run Pulse or Java UDX functions until the session currently running Pulse functions is closed.

While resource pool settings are based on the resources of a node, they apply across the entire cluster. A session with a Vertica Pulse query reserves the same resources for its JVM on all nodes in the cluster. Therefore, it doesn't matter if the cluster contains 3 nodes or 30 nodes;

each node reserves, for example, 4GB of the node's memory for the JVM used by the Vertica Pulse session and `PLANNEDCONCURRENCY` limits the amount of sessions that can run Pulse cluster-wide. If `PLANNEDCONCURRENCY` is 1, then only 1 vsql session (or client connection) in the entire cluster can run Pulse.

You can display the current resource pool settings for the `jvm` resource pool with the following command:

```
select name, MAXMEMORYSIZE, PLANNEDCONCURRENCY from V_CATALOG.RESOURCE_POOLS
where name = 'jvm';
```

Configuring the `jvm` Resource Pool for your System

Do not use the default `jvm` resource pool settings for Vertica Pulse. You must configure the `jvm` resource pool to match your hardware and workload requirements. Specifically, specify `PLANNEDCONCURRENCY` and `MAXMEMORYSIZE` to match your hardware.

You may need to experiment to find the optimal settings for your hardware and your specific workloads. As a best practice, allow:

- At least 2GB of memory per session for Vertica Pulse
- At least 25% of the memory available for general Vertica overhead. Essentially, `MAXMEMORYSIZE` must never exceed 75% of total system memory.

Note: If you are running a lot of queries not in the context of Vertica Pulse, then you should allow for more memory to be available outside of the `jvm` resource pool.

To configure your system for Vertica Pulse:

- Determine the number of cores on a node. Your `PLANNEDCONCURRENCY` setting cannot exceed this value. For example, you can run the following from a shell to determine cores:

```
cat /proc/cpuinfo | egrep "core id|physical id" | tr -d "\n" | sed s/physical/\\nphysical/g |
grep -v ^$ | sort | uniq | wc -l
```

- Determine the amount of memory in GB on a node. Your `MAXMEMORYSIZE` cannot exceed 75% of the total system memory. For example, you can run the following from a shell to determine the Total System Memory in GB for any particular node:

```
awk /MemTotal/{printf "%f GB\n", $2/1024/1024}' /proc/meminfo
```

- Use the formula $MAXMEMORYSIZE / PLANNEDCONCURRENCY$ to determine how much memory each Vertica Pulse JVM receives. For example, you can use $(.75 * Total\ System\ Memory) / PLANNEDCONCURRENCY$ if you plan to use most of your RAM for Vertica Pulse. The outcome of the formula must be 2 (which denotes GB) or greater. For example, if you have 8GB of total system memory, and your estimated `PLANNEDCONCURRENCY` is 3, then the formula results in "2" and is acceptable. However, if you have the same amount of memory and `PLANNEDCONCURRENCY` is set to 4, then the result of the formula is "1.5", which is below the recommended minimum of 2GB. You can either add more RAM to the system or reduce `PLANNEDCONCURRENCY` to get the resulting number up to "2".
- Finally, alter the `jvm` resource pool. For example, for a cluster with nodes each having 16GB of memory, and you determine to use up to 75% of the total system memory ($0.75 * 16GB = 12GB$) for Vertica Pulse, then you can set the resource pool as follows:

```
ALTER RESOURCE POOL jvm MAXMEMORYSIZE '12G' PLANNEDCONCURRENCY 3;
```

Note: For evaluation purposes on systems with lower memory, set `MAXMEMORYSIZE` to 75% and `PLANNEDCONCURRENCY` to 1: `ALTER RESOURCE POOL jvm MAXMEMORYSIZE '75%' PLANNEDCONCURRENCY 1;` While these settings are unsupported, they do allow you to run simple Vertica Pulse queries. You may experience Out Of Memory exceptions and slow performance.

For additional details, see:

- [ALTER RESOURCE POOL](#)
- [Managing Workloads](#)
- [Java UDx Resource Management](#)

Assign Users to the `pulse_users` Role and Allow Access to Pulse Functions

When you install Pulse, the install script creates a pulse schema, which contains the user-dictionary and mapping lists used by Pulse. Initially only administrators can read or edit tables in the pulse schema. To give non-administrator database users access to the pulse schema, you assign the user to the 'pulse_users' role, which has all privileges for the pulse schema. The role is created automatically when you install Pulse.

Note: The default dbadmin user has access to the pulse schema by default. You do not need to add the `pulse_users` role to the dbadmin account.

Granting users Access to the Pulse Schema

To grant non administrator users access to the tables in the Pulse schema:

1. As the dbadmin, if the user does not exist, create the user with the command: `create user username identified by 'password';`
2. As the dbadmin, if the user does not have access to function in the `public` schema, then grant execute privileges with the command: `GRANT execute ON ALL FUNCTIONS IN SCHEMA public TO username;`

Note: By default, the Pulse functions are created in the `public` schema.

3. As the dbadmin, grant the `pulse_user` role to the new user with the command: `grant pulse_users to username;`
4. As the user to which you granted the `pulse_user` role, set the users role to `pulse_users` with the command: `set role pulse_users;`

Note: The user must run the `set role` command per session to read or edit tables in the pulse schema.

Uninstalling Vertica Pulse

Uninstalling Vertica Pulse on hosts and uninstalling Pulse packages require different procedures.

Uninstall Vertica Pulse on Your Hosts

As the dbadmin, run the uninstall script from the node on which you installed the Pulse package:

```
bash /opt/vertica/packages/pulse/uninstall.sh
```

The uninstall script removes all Pulse functions, but does not remove the pulse schema containing the user-dictionary and mapping tables.

To remove all Pulse dictionaries and mappings, including custom dictionaries, include the `-r` parameter

```
bash /opt/vertica/packages/pulse/uninstall.sh -r
```

Uninstall Pulse Packages

To uninstall the Pulse package, on the nodes that have the Pulse package installed, use the appropriate command for your package.

- For RPM packages:

```
# sudo rpm -e vertica-pulse
```

- For DEB packages:

```
# sudo dpkg --remove vertica-pulse
```

The Pulse schema and associated user-dictionary and mapping tables remain in the database. To remove the Pulse schema and its associated tables, run the following command:

```
DROP SCHEMA pulse CASCADE
```

Using Pulse

Dictionaries and Mappings

Pulse contains built-in dictionaries and maps that help determine the sentiment of sentences. You have the option of creating and loading user-defined dictionaries and maps.

Dictionaries and Mappings are loaded across all client sessions and remain in memory even if the database is stopped and started.

Dictionaries

Pulse uses a proprietary system dictionary to help score sentiment. The system dictionary is not visible or modifiable. You can, however, alter the default way that Pulse scores sentiment by modifying user dictionaries. The user dictionaries provide flexibility so that you can tune sentiment scoring for your specific domain. You do not have to modify user dictionaries if Pulse is scoring your data appropriately.

Users can apply dictionaries on a per-user basis. Any number of Pulse users can concurrently apply different sets of dictionaries without conflicts and without disrupting the sessions of other users. Each user can have one dictionary of each type loaded at any given time. If a user does not specify a dictionary of a given type, Pulse uses the default dictionary for that type.

Mappings

Maps are lists of synonyms of one or more words that map to another word. Using maps allows you to analyze text that pertains to the same subject or concept but may use slightly different terminology.

For example, you can map both 'Hewlett Packard' and 'Hewlett-Packard' (with hyphen) to 'HP.' Pulse substitutes the mapped words to the core word when it runs its analysis.

Dictionary and Mapping Tables

User dictionaries and a normalization map for each supported language reside in tables inside the Pulse schema. You can see the contents of the tables with simple queries such as:

```
SELECT * FROM pulse.pos_words_en;
```

Or:

```
SELECT * FROM pulse.pos_words_es;
```

There is one table per dictionary/map for each language. The table name has the language abbreviation as a suffix. For example, English tables have the suffix "_en" and Spanish tables have the suffix "_es". By default, the user dictionaries and normalization map are empty. You can modify these tables to tune Pulse to your specific needs. After you modify these tables, you must load the changes into memory.

You can update the user dictionaries and normalization tables at any time. To do so, you must run load functions (see [LoadDictionary\(\)](#) and [LoadMapping\(\)](#)) to load the values from the tables into memory. Your changes affect sentiment scoring only after you load the new values.

Note: Loading a user dictionary or loading a normalization map overwrites the values in memory with the values from the specified table. You cannot append user dictionaries or the normalization map in memory.

The following dictionary table names provide descriptions of the English user dictionaries. For Pulse versions that support Spanish, the same set of dictionaries with the suffix "_es" is present in the Pulse schema.

Dictionary Table Name	Description
white_list_en	Words that are always marked as an attribute. This list augments the built-in Pulse attribute discovery process. Add words that you always want scored to the white_list user dictionary. For example, such words can include nouns, phrases or business-dependent attributes that are not auto-discovered by Pulse. This list is typically modified to increase the accuracy of sentiment scoring for your domain.
stop_words_en	Words that are never marked as an attribute. Add words that you do not want scored to the stop_words user dictionary. Use this dictionary to filter out attributes that are not of interest to your analysis. This list is typically modified to increase the accuracy of sentiment scoring for your domain. The stop_words dictionary can only contain nouns and compound nouns. If Pulse does not identify a stop word as a noun, it ignores it.
pos_words_en	Positive words that can be any type of word or phrase. Words in this list are more likely to carry a positive polarity in general. You can also add exact phrases, such as idioms, to this list.

Dictionary Table Name	Description
	Examples: <i>adroit, resolve, strong, hit the nail on the head</i>
neg_words_en	Negative words that can be any type of word or phrase that have a negative connotation. Words in this list are deemed more likely to carry a negative polarity in general. You can also add exact phrases, such as idioms, to this list. Examples: <i>abhorrent, butcher, racist, wrath, flash in the pan.</i>
neutral_words_en	Words that indicate a neutral connotation. Words in this list are scored with a sentiment of 0, meaning not positive or negative.

The following table shows the tables that describe mapping within Pulse.

Mapping Table Name	Description	Example
normalization_en	A list of word pairs used to map like terms (synonyms). You can use this to correct common misspellings and map them to the correct spelling. This list is frequently modified and is empty by default.	base/synonym: <ul style="list-style-type: none"> • 'hp'/ 'hewlett-packard' • 'hp'/ 'Hewlett-Packard' • 'Obama'/ 'President Obama' • 'Obama'/ 'Barack Obama'

Loading Dictionaries and Mappings into Pulse

If you have made changes to the Pulse schema tables, then you must load either the dictionaries, the normalization map, or both. After the changes are loaded, Pulse stores them in memory across all sessions in the cluster. Because Pulse automatically loads the dictionaries and mapping at startup, you do not need to reload them after a database restart or system reboot.

- To load an individual user dictionary into memory, use the [LoadDictionary\(\)](#) function with the appropriate parameter and word list.
 - `LoadDictionary` does not append user-dictionary lists. Instead, it overwrites them. If you load a user dictionary more than once with the same list name, then only the most recent user dictionary is loaded for that list name.
- To load the normalization mapping into memory, use the [LoadMapping\(\)](#) function with the normalization map.
 - If you load a mapping with an incorrect `mapName`, then the result of `LoadMapping()` is false and the map is not loaded. `LoadMapping()` does not append maps. Instead, it overwrites them. If you load a map more than once with the same `mapName`, then only the most recent mapping is loaded for that `mapName`.
 - If `LoadMapping()` is successful, Vertica returns a success message from each node in the cluster.

Automatically Loading Dictionaries and the Normalization Map

For ease of use, Pulse ships with a script to automatically load into memory all of the required user dictionaries and the normalization mapping. This script only exists on the node on which you installed the Pulse RPM/DEB package.

You can run the script from within `vsq` with the following command:

```
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
```

Manually Loading Dictionaries and the Normalization Map

If you want to manually load certain user dictionaries or mappings from the Pulse schema tables, run the following command. This example loads the `pos_words` dictionary. See [LoadDictionary\(\)](#) for valid values for the `listName` parameter and for multilingual version loading.

Note: The following examples use the English dictionaries. For Spanish, replace `"_en"` with `"_es"`.

1. Add a word to the pos_words dictionary:

```
=> INSERT INTO pulse.pos_words_en VALUES('SuperDuper');  
=> COMMIT;
```

By default, added words are not case sensitive. "ERROR" produces the same results as "error". You can, however, specify a case setting for a single word using the \$Case parameter. For example, to identify "Apple", rather than "apple", you would add the following:

```
=> INSERT INTO pulse.white_list_en VALUES('$Case(Apple)');  
=> COMMIT;
```

2. Load the updated dictionary into Pulse:

```
=> SELECT LoadDictionary(standard USING PARAMETERS  
listName='white_list') OVER()  
FROM pulse.white_list_en;
```

3. If you change the normalization map, you can load the new normalization values with the following command:

```
=> SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS  
mapName='normalization') OVER() FROM pulse.normalization_en;
```

After loading, Vertica returns a success message and the number of rows (words or word pairs) loaded.

Dictionary and Mapping Labels

You can apply a label to any user-defined dictionary or mapping when you load that object. Labels enable you to perform sentiment analysis against a predetermined set of dictionaries and mappings without having to specify a list of dictionaries. For example, you might have a set of dictionaries labeled "music" and a set labeled "movies." The default user dictionaries automatically have a label of "default."

A single dictionary or mapping can have multiple labels. For example, you might label a white list of artists as both "painters" and "renaissance." You could load the dictionary by loading either label. A label can only apply to one dictionary of each type. For example, you cannot have two dictionaries of stop words that share the same label. If you apply a label to multiple dictionaries of the same type, Pulse uses the most recently applied label.

You can view the labels associated with your current dictionaries using the [GetAllLoadedDictionaries\(\)](#) function. You can also view the label associated with your current mapping using the [GetLoadedMapping\(\)](#) function.

Normalization Map Effect on Results

Before any of the sentiment analysis functions are run on the text, the normalization map is applied. When a sentiment analysis function is run, Pulse replaces the synonym with the base word. The result of the sentiment analysis function displays the mapped words and not the original text. For example, Pulse maps both 'Hewlett Packard' and 'Hewlett-Packard' (with a hyphen) to 'HP' in the results when the normalization map is populated with those terms.

Before Mapping

The following example demonstrates sentiment analysis before mapping:

```
=> SELECT SentimentAnalysis('Hewlett-Packard was founded in 1939.  
Hewlett Packard was started in a garage in Palo Alto California')  
OVER(PARTITION BEST);
```

sentence	attribute	sentiment_score
1	hewlett-packard	0
2	hewlett packard	0
2	garage	0
2	palo alto california	0

(4 rows)

Insert Normalization Values and Load Map

You can add values to the normalization map using an INSERT statement. The following example demonstrates how to insert normalization values and load the map:

```
=> INSERT INTO pulse.normalization_en VALUES('HP', 'Hewlett-Packard');  
=> INSERT INTO pulse.normalization_en VALUES('HP', 'Hewlett Packard');  
=> COMMIT;  
  
=> SELECT LoadMapping(standard_base, standard_synonym  
USING PARAMETERS mapName='normalization') OVER()  
FROM pulse.normalization_en;
```

You can also map multiple values to the same term using a `$LIST` parameter. The following example would map multiple alternate names for the city of Boston to the value 'Boston'.

```
INSERT INTO normalization_en Values( 'Boston', '$LIST(BOS,beantown,the hub);
```

After Mapping

The mapping operation replaces the attributes with their counterparts from the normalization list and displays the base terms:

```
=> SELECT SentimentAnalysis('Hewlett-Packard was founded in 1939.  
Hewlett Packard was started in a garage in Palo Alto California')  
OVER(PARTITION BEST);
```

sentence	attribute	sentiment_score
1	hp	0
2	hp	0
2	garage	0
2	palo alto california	0

(4 rows)

The `CommentAttribute()` function also uses the normalization map and displays the base terms instead of the original text:

```
=> SELECT CommentAttributes('Hewlett-Packard was founded in 1939.  
Hewlett Packard was started in a garage in Palo Alto California')  
OVER(PARTITION BEST);
```

sentence	attribute
1	hp
2	hp
2	garage
2	palo alto california

(4 rows)

Creating Tables for Custom Dictionary Mappings

The Vertica Pulse package includes all the necessary user dictionary and mappings tables. However, you can create your own tables to store additional user dictionaries or mappings. For example:

```
CREATE TABLE my_positive_words(word VARCHAR(64));
```

The following example shows how to create a table, add terms to it, and then load the table as a normalization map:

```
=> CREATE TABLE myNormalization(base VARCHAR(64), synonym VARCHAR(64));  
=> INSERT INTO myNormalization VALUES('hp', 'Hewlett Packard');  
=> INSERT INTO myNormalization VALUES('hp', 'Hewlett-Packard');  
=> COMMIT;  
  
=> SELECT LoadMapping(base, synonym USING PARAMETERS
```

```
mapName='normalization') OVER() FROM myNormalization;
```

After loading, Vertica returns a success message from each node in the cluster.

Using Action Patterns in Dictionaries

Vertica Pulse supports the use of action patterns in `white_list` dictionaries only. An action pattern enables Pulse to recognize phrases that denote action, intention, or interest, such as *going to buy*, *waiting to see*, and so on. Action patterns can identify behaviors associated with your sentiment analysis terms.

Action patterns can:

- **Connect Word Forms to a Root Word** — Vertica Pulse lemmatizes all words. *Lemmatization* recognizes different word forms and maps them to the root word. For example, Pulse would map *bought* and *buying* to *buy*. This ability extends to misspellings. For example, *tryiiing* and *seeeeee taaablets* would map to *trying* and *seeing tablets*.
- **Create Object-Specific Queries** — To identify only the attributes that are objects of action patterns, create a [whitelist dictionary](#) that contains only action patterns of interest. In your sentiment analysis query set the `actionPattern` and `whiteListOnly` parameters to `true`.

Note: Action patterns exist in the whitelist dictionary. If a word that matches an action pattern appears in both the `white_list` and `stop_words` dictionaries, the `white_list` takes precedence. The `stop_list` word would appear in sentiment analysis results.

Action Pattern Syntax

Construct an action pattern by combining action parameters within an `#action`. By default, parameters match any instance of the associated part of speech. You can match specific terms by listing them with the parameter. For example, the parameter `$PREP(to, on)`, would match only *to* and *on*.

Parameters can also accept [\\$regex](#) and [\\$list](#) operators.

```
#action{$ADV $VERB $PREP $ADJ}
```

Parameter	Short form	Description
<code>\$ADJECTIVE</code>	<code>\$ADJ</code>	Matches any adjective.

\$ADVERB	\$ADV	Matches any adverb.
\$PREPOSITION	\$PREP	Matches any preposition.
\$VERB	\$VERB	Matches any verb.

Default Action Patterns

Pulse includes default action patterns in the whitelist dictionary. You cannot remove these patterns. Pulse always evaluates them when you perform a sentiment analysis.

Language	Pattern	Example
English	\$Verb \$Prep \$Verb	<i>planning on buying, thinking about dropping</i>
	\$Verb TO \$Verb	<i>going to buy, looking to acquire</i>
Spanish	\$Verb \$Prep \$Verb	<i>voy a comprar, pienso en dejar</i>
	\$Verb \$Verb	<i>quiero solicitar, planean adquirir</i>

Examples

The following example shows how Pulse can match *customer* or *client*, the verb *would* and any other verb. It would match phrases like *customer would buy* or *client would cancel*.

```
INSERT INTO pulse.white_list_en values('#action{${LIST(customer,client)} would $VERB}');
```

The following example shows a match for specific verbs *like*, *want*, and *plan*, plus any preposition and any other verb. It would match phrases like *want to own* or *plan on buying*.

```
INSERT INTO pulse.white_list_en values('#action{${VERB(like,want,plan)} $PREP $VERB}');
```

The following example identifies words ending in *ember*, such as *December*, and uses a regular expression to identify date references, such as *2nd* or *4th*. This action pattern could identify users planning to attend an event or making holiday plans.

```
INSERT INTO pulse.white_list_en values('#action{On $regex(.*ember) $regex(\d+(th|st|nd|nd)) I will $verb to}');
```

Using Lists In Dictionaries

Vertica Pulse supports the use of token-based lists in dictionaries. You can use a list to match multiple terms to a single word. Token based lists differ from mapping by allowing you to create multiple associations in a single action rather than a series of pairs. Unlike mapping, lists are not restricted to the normalization dictionary.

Note: Token-based lists do not apply to the base word in normalization dictionaries.

You can add lists to the following dictionaries:

- pos_words
- neg_words
- neutral_words
- normalization
- white_list
- stop_words

You can add a token to a user-defined dictionary using an INSERT statement and \$LIST parameter containing the list of values to match.

The following example would match slang terms to the word "good".

```
INSERT INTO pos_words Values( 'good', '$LIST(sweet,dope,tight);
```

Using Regular Expressions in Dictionaries

Vertica Pulse supports the use of regular expressions in [user-defined dictionaries](#). Vertica Pulse regular expressions use the [java.util.regex](#) package syntax. For more information on this syntax, refer to the Oracle documentation.

Note: Regular expressions do not apply to the base word in normalization dictionaries.

You can add regular expressions to the following dictionaries:

- pos_words
- neg_words
- neutral_words
- normalization
- white_list
- stop_words

You can add a regular expression to a user-defined dictionary using an [INSERT](#) statement and \$REGEX parameter containing the regular expression. Regular expressions are case insensitive. the regular expression \$regex(apple) produces the same matches as the regular expression \$regex(Apple).

Note: A regular expression can support a single token or word. *Smartphone* would be a valid token, but *smart phone* would not .

The following example would match any word ending with the string "day". You could use it to identify the days of the week or words such as *yesterday* and *today*.

```
INSERT INTO stopwords_en Values( '$LIST(nice,good,fine) $REGEX(.*day)');
```

The following example matches references to iPhones, including the number and letter version.

```
INSERT INTO whitelist_en Values('Iphone $REGEX(\d{1}(S|C)?)');
```

To use a parenthesis as a literal part of a regular expression, you must use the escape character \ twice to prevent Pulse from interpreting the parenthesis as metacharacter in the regular expression. The following example would match the literal string (hugs).

```
INSERT INTO whitelist_es Values($REGEX(\\(hugs\\));
```

Determining Sentiment

You determine sentiment by using the [SentimentAnalysis\(\)](#) function on text.

The [SentimentAnalysis\(\)](#) function first extracts the attributes (typically nouns) from the sentence, and then applies a sentiment score to each attribute. Scores can be one of the following:

- **1** - Positive Sentiment
- **0** - Neutral Sentiment
- **-1** - Negative Sentiment

This provides a more granular analysis than just determining the sentiment for the sentence as a whole. Consider the following quote from Abraham Lincoln; "*Force is all-conquering, but its victories are short-lived.*" If you were to score the sentiment of the sentence as a whole by averaging the sentiment of its parts, then the sentiment is neutral.

```
=> select avg(t1.sentiment_score) as 'Average Sentiment' from (  
    select sentimentAnalysis('Force is all-conquering, but its victories are short-lived.')  
    over (PARTITION BEST)  
    ) as t1;  
  
Average Sentiment  
-----  
0
```

If you score the individual attributes of the sentence, then you can obtain a much more precise analysis of the sentiment than if you were trying to assign a single score to the entire sentence. For example:

```
=> select sentimentAnalysis('Force is all-conquering, but its victories are short-lived.') over  
(PARTITION BEST);  
  
sentence | attribute | sentiment_score  
-----+-----+-----  
1 | force | 1  
1 | victories | -1
```

"Force" is scored with positive sentiment because it is "all-conquering". "Victories" is scored with negative sentiment because it is "short-lived".

Note: Vertica Pulse does not recognize personal pronouns (I, you, we, he, she, it, etc.) as attributes.

`SentimentAnalysis()` also extracts the sentiment from multiple sentences and returns the sentence in which attributes are found:

```
=> SELECT SentimentAnalysis('Force is all-conquering, but its victories are short-lived. Every good boy deserves fudge.') OVER(PARTITION BEST);
```

sentence	attribute	sentiment_score
1	force	1
1	victories	-1
2	boy	1
2	fudge	1

(4 rows)

"Boy" is scored with positive sentiment because he is good. Fudge is scored with positive sentiment because it is something that is deserved.

Note: The sentence detector considers a period to mark the end of a sentence. Some abbreviations that use a period, such as Dr. or Mr., cause the sentence detector to end the sentence at the abbreviation.

The `SentimentAnalysis` function also identifies attributes with neutral sentiment (a sentiment score of zero). For example:

```
SELECT SentimentAnalysis('Roses are red. Violets are blue.') OVER(PARTITION BEST);
```

sentence	attribute	sentiment score
1	roses	0
2	violets	0

(2 rows)

Both roses and violets receive neutral sentiment because neither being red nor blue is considered positive or negative in this context.

See the [Pulse Cookbook](#) for more examples of determining sentiment.

Sentiment Analysis Levels

Vertica Pulse is capable of determining sentiment at the following levels:

- Attribute
- Sentence
- Document

You can specify an analysis level using the granularity parameter of the `SentimentAnalysis` function. You can perform multiple levels of analysis simultaneously within the same query.

Attribute-Level Analysis

Attribute level analysis provides a sentiment for each object in a sentence. This behavior is the default level of analysis for Pulse. Attribute analysis identifies the objects of a sentence and any sentiment expressed regarding those objects.

The following example shows the sentiment expressed with regard to "camera" and "quality pictures."

```
Select SentimentAnalysis ('The camera takes great quality pictures but is expensive. It feels like a professional one' USING PARAMETERS
granularity='A') over();
sentence | attribute | sentiment_score
-----+-----+-----
1 | camera | 1
1 | quality pictures | 1
```

Sentence-Level Analysis

A sentence level analysis provides the overall sentiment of each sentence in a document. If a sentence contains both positive and negative sentiments, it appears as mixed.

The following example shows two sentences, the first of which is mixed. As a mixed sentiment, the sentiment score is 0, or neutral, and the mixed value is true. The second sentence is entirely positive. Its sentiment is 1, or positive, and the mixed value is false.

```
Select SentimentAnalysis ('The camera takes great quality pictures but is expensive. It feels like a professional one' USING PARAMETERS
granularity='S') over();
sentence | sentiment_score | mixed
-----+-----+-----
1 | 0 | true
2 | 1 | false
```

Document-Level Analysis

Document level analysis provides the overall sentiment of an entire document. If you wanted to know if a movie review was positive, negative, or mixed, a document level analysis could provide that information. Document level analysis gives both the overall sentiment score and a mixed rating if the sentiment is not exclusively positive or negative.

The following example shows that overall, the writer is positive but does express some negative sentiments.

```
Select SentimentAnalysis ('The camera takes great quality pictures but is expensive. It feels like a professional one' USING PARAMETERS  
granularity='D') over();  
sentiment_score | mixed  
-----+-----  
1 | true
```

Tuning Pulse

Pulse contains built-in dictionaries that help to determine the sentiment of sentences. These dictionaries are not directly readable. However, you can modify the Pulse dictionary tables to improve automatic attribute discovery and provide more accurate results for sentiment scoring based on your specific data sets. The dictionary tables are available in the Pulse schema. Any words you add to these dictionaries takes precedence over the built-in dictionaries.

Improving Automatic Attribute Discovery

Pulse identifies nouns in sentences and marks them as attributes. However, there are two dictionaries and one mapping that you can modify to improve automatic attribute discovery. These are:

- `white_list` - a list of words on which you want to score sentiment, but are not auto-discovered by Pulse. Typically these are product or company names, or special words in the domain of the data you are analyzing. You can also add noun phrases to the `white_list`.
 - Consider the term "President Smith". Pulse automatically marks "President" as an attribute. However, you can add "President Smith" to the `white_list` and Pulse then uses "President Smith" as the attribute instead of just "President".
 - If your `white_list` contains phrases that are subsets of other phrases in the white list, then the shorter phrase is not matched if the text being analyzed matches the superset phrase. For example, if both "Honest AI" and "Honest AI Car Emporium" are in the `white_list`, then the latter phrase is identified as an attribute in the text "Honest AI Car Emporium is not honest.", not the shorter "Honest AI" `white_list` phrase.
- `stop_words` - a list of words on which you do not want to score sentiment, but may appear frequently in your data set. `stop_words` is basically a way to filter out attributes.
 - If a word appears in both `stop_words` and `white_list`, then the `white_list` word takes precedence. The word appears in results even though it is in the `stop_words` dictionary.
- `normalization` - a map of base words and synonyms that allow you to normalize words for easy comparison. For example, you can normalize "Hewlett Packard" to "HP", then count the number of times "HP" appears as an attribute in your data. Any text that contains "HP" or "Hewlett Packard" is counted towards the total.

Determining How Pulse Scores Sentiment

When tuning Pulse it is important to understand why Pulse may not be scoring a particular attribute the way you want it to be scored. For example, consider the sentence "The quick brown fox jumped over the lazy dog." By default, Pulse scores the fox as positive and the dog as negative. If you want to better understand how the words in the sentence affect the attributes, then you can use the *relatedwords* parameter to see which words are affecting the score. For example:

```
select SentimentAnalysis('The quick brown fox jumped over the lazy dog.'  
  USING PARAMETERS relatedwords=true) OVER(PARTITION BEST);
```

sentence	attribute	sentiment_score	related_word_1	related_word_2	related_word_3
1	fox	1	quick	lazy	
1	dog	-1	lazy		

(2 rows)

The output details that "quick" and "lazy" impacted the scoring of the "fox" attribute, and that "lazy" affected the scoring of the "dog" attribute. "Quick" (positive) is weighted heavier than "lazy" (negative) when scoring "fox" because the word "quick" is closer to the attribute "fox" in the sentence, and the result is that "fox" is scored positively. "Lazy" (negative) is the only related word being used to score the sentiment for "dog". If you don't agree with the scoring, you can change how these related words affect the score by adding them to the appropriate user-dictionary, as described in "Improving Sentiment Scores".

Improving Sentiment Scores

Pulse scores sentiment on attributes (nouns) in sentences using Natural Language Processing (NLP) algorithms and rules. Pulse attempts to identify the parts of a sentence (for example, verbs, nouns/attributes, adjectives, etc; the parts of speech), and then scores the attributes based on which system-dictionaries the parts of speech appear (positive, negative, or neutral) and where those parts of speech appear in relation to the attributes and other contextual information. Pulse does not identify personal pronouns (he, you, we, she, etc.) as attributes.

Pulse provides a [PartsOfSpeech](#) function so that you can verify which parts of speech are being identified in a sentence.

Sentiment Scoring and the Precedence of Pulse User-Dictionaries

The negative, positive, and neutral user-dictionaries adjust the score of an attribute based on which dictionary the words in the sentence appear. User-dictionaries take precedence over the internal dictionaries that Pulse uses for analyzing text, so that you can override the default polarity of an opinion word or phrase by inserting it in the appropriate user-dictionary table.

Pulse also supports using phrases in the `pos_words`, `neg_words` and `neutral_words` dictionaries. Phrases, such as idioms ("hit the nail on the head."), can be added to the user dictionaries. Phrases of two or more words support "fuzzy" matching. For example, the phrase "solve problem" also matches "solves problems".

Pulse uses an order of precedence to determine which user dictionary is used to modify the default score. The order of precedence of the user dictionary that Pulse uses to score attributes is as follows:

1. Phrases or strings that occur in the "neutral_words" dictionary
2. Phrases or strings that occur in the "neg_words" dictionary
3. Phrases or strings that occur in the "pos_words" dictionary
4. Single words appearing in the "neutral_words" dictionary
5. Single words appearing in the "neg_words" dictionary
6. Single words appearing in the "pos_words" dictionary

Note: If a word is present in both `stop_words` and `white_list`, then the `white_list` word takes precedence. The word is present in results even though it exists in `stop_words`.

Consider the sentence "Fudge is good". It contains three parts; a noun (fudge), a verb (is), and an adjective (good). When you analyze the sentence using Pulse, it identifies "fudge" as an attribute, because it is a proper noun, and then assigns "fudge" a positive sentiment:

```
select sentimentAnalysis('Fudge is good') OVER(PARTITION BEST);
  sentence | attribute | sentiment_score
-----+-----+-----
1 | fudge | 1
```

The number of words matched against a dictionary also has an impact on which dictionaries take precedence. For example, phrases or word combinations in the user-dictionary lists take precedence over single words. For example, the positive phrase "solve problem" causes a positive score on the text "Joe solves problems", even though "problem" is a negative word. Since phrases have precedence over single words, a positive score is applied to Joe.

```
SELECT SentimentAnalysis('Joe solves problems.') OVER(PARTITION BEST);
```

```
sentence | attribute | sentiment_score  
-----+-----  
1 | joe      | 1  
(1 row)
```

```
SELECT SentimentAnalysis('Joe is a problem.') OVER(PARTITION BEST);
```

```
sentence | attribute | sentiment_score  
-----+-----  
1 | joe      | -1  
1 | problem  | 0  
(2 rows)
```

Tuning Example

You can modify any of the user-dictionaries to improve the accuracy of sentiment scores. The two basic dictionaries, "neg_words" and "pos_words", are typically the easiest to modify to get good results. Words in these two dictionaries can be any part of speech (verb, adjective, etc.). If you find a word that is causing an attribute to be scored positively or negatively, but it should be scored as neutral, then you can add that word to the "neutral_words_en" dictionary to cause it to be scored 0.

Consider the sentence "The product delivers simplicity.":

```
select sentimentAnalysis('The product delivers simplicity.') over(PARTITION BEST);
```

```
sentence | attribute | sentiment_score  
-----+-----  
1 | product  | 0  
1 | simplicity | 0  
(2 rows)
```

If you want "product" to be scored positively in this sentence, then you must add "deliver simplicity" to the pos_words user-dictionary. "deliver simplicity" will also match "delivers simplicity" due to the "fuzzy" matching feature of phrases in the dictionaries. If you add "simplicity" by itself to the "pos_words" dictionary, then simplicity in any context is considered positive, which may not be the result you want to achieve across your entire domain. The following example adds "deliver simplicity" to the pos_words user-dictionary for the English language:

```
insert into pulse.pos_words_en values ('deliver simplicity');
commit;

-- you must reload the dictionaries for the changes to be effective
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql

select sentimentAnalysis('The product delivers simplicity.') over(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | product   |                 1
(1 row)
```

Notice that "simplicity" is not positive if it is not paired with "deliver":

```
select sentimentAnalysis('The product provides simplicity.') over(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | product   |                 0
          1 | simplicity |                 0
(2 rows)
```

If you want "simplicity" to always be positive, add it to the "pos_words" list. This example replaces "deliver" with "provides":

```
insert into pulse.pos_words_en values ('simplicity');
commit;

\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql

select sentimentAnalysis('The product provides simplicity.') over(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | product   |                 1
          1 | simplicity |                 0
(2 rows)
```

Notice that the sentiment score for the attribute (noun) "simplicity" is not affected by having the word "simplicity" in a Pulse user-dictionary, since it has been identified as an attribute.

Additional Tuning Examples

The following table provides additional examples for tuning Pulse:.

Text	Attribute	Score	Tuning Steps
New product smashes kickstarter target in a day!	New Product	Default: -1 After Tuning: 1	"Smash" is scored negatively by default. Add "smash

Text	Attribute	Score	Tuning Steps
			target" to "pos_words".
Get a sneak peek of the new movie.	Movie	Default: -1 After Tuning: 1	"sneak" is scored negatively by default. Add "sneak peek" to "pos_words".
Google was able to spot trends in flu outbreaks in the United States using the collection and analysis of big data.	Google	Default: -1 After Tuning: 1	"outbreak" is scored negatively by default. Add "spot trend" to "pos_words".
Five health tips that will knock your socks off!	health tips	Default: -1 After tuning: 1	"knock" is scored negatively by default. Add "knock your socks off" to "pos_words".

If you have many words or base/synonyms to add to user-dictionaries, then you can bulk load the lists from text files. See [Bulk Loading Word Lists from Text Files](#).

Bulk Loading Word Lists from Text Files

If you have many words that you need to add to the user-dictionary or normalization mapping, then it may be easier to create the word lists in a text file and load the lists using the COPY command.

Bulk Loading User Dictionary Lists

To bulk load user-dictionary lists into the pulse schema, first create a text file with the list of words to add, one word per line, for each of the user-dictionaries. See [Dictionaries and Mappings](#) for a list of the user-dictionaries and normalization map. Optionally name each text file to match the name of the corresponding user-dictionary. Place these text files in the `/home/dbadmin` directory.

Then, in vsql, use one or more of the following commands to load the respective text file into the pulse schema. These commands assume that you are using English version of Pulse, that the built-in user dictionary tables in the pulse schema and that the text files are named the same as the user-dictionary.

```
copy pulse.neg_words_en(standard) from '/home/dbadmin/neg_words.txt';
copy pulse.neutral_words_en(standard) from '/home/dbadmin/neutral_words.txt';
copy pulse.pos_words_en(standard) from '/home/dbadmin/positive_words.txt';
copy pulse.stop_words_en(standard) from '/home/dbadmin/stop_words.txt';
copy pulse.white_list_en(standard) from '/home/dbadmin/white_list.txt';
```

Bulk Loading the Normalization Map

You can load normalization terms into the pulse schema similarly to loading user-dictionaries. However, instead of one word per line, use the convention of one pair of words per line, separated by a comma. For example, to map the different forms of Micro Focus to HP, create a text file in `/home/dbadmin` named `normalization.txt` with the following content:

```
hp, hewlett packard
hp, hewlett-packard
```

Then, in vsql, use the following command to load the normalization into the pulse schema.

```
copy pulse.normalization_en (standard_base, standard_synonym) from '/home/dbadmin/normalization.txt'
delimiter ',';
```

When you have finished loading the text files, run the `loadUserDictionaries.sql` script to update the new terms in memory:

```
vsql -f /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
```

Multilingual Pulse

This section describes the multilingual features of Pulse and gives a brief explanation on how to use the `sentimentAnalysis()` functions for different supported languages.

Pulse can analyze text in different languages. Currently English and Spanish are supported. You can specify the language that is analyzed in three ways:

- Provide the language as argument: if there is a language specified in the document record, then it can be used for analyzing the text by passing it as argument. This is particularly useful when a dataset contains texts in different languages. If the language in a record is not a supported one, then it is ignored.
- Provide the language as parameter: if there is no value specified for the language for a document record, Pulse uses the value specified for the language parameter in the query to get the language.

Note: If you provide the language parameter more than once, then the last value specified is used.

- Do not provide an argument or parameter and use the default language. If the language is neither specified in the record nor by the user, then Pulse defaults to English unless you have changed the default language. To change the default language use the [SetDefaultLanguage](#) function.

Note: If you provide both an argument and a parameter, then the argument is used as the language. If the argument is not valid then the parameter is used. If neither the argument or parameter are valid then the default language is used.

Note: Accents are removed from characters in attributes. Additionally, a "u" with a dieresis is converted to a plain "u" and an "n" with a diacritical tilde is replace with a plain "n".

Functions that use language as parameter and/or as argument:

- [CommentAttributes](#)
- [ExtractSentence](#)
- [GetAllSentences](#)
- [GetSentenceCount](#)

- [PartsOfSpeech](#)
- [SentimentAnalysis](#)

Other functions can use the language only as a parameter (if not provided, the function uses the default language):

- [GetLoadedDictionary](#)
- [GetLoadedMapping](#)
- [LoadDictionary](#)
- [LoadMapping](#)
- [GetAllDictionaryWords](#)
- [GetAllMappingWords](#)

In This Section

Spanish Pulse

The only visible difference between the English and Spanish versions is in the table names for the user dictionaries. The modifications for dictionaries/mappings must be done in the following tables:

- `white_list_es`
- `stop_words_es`
- `pos_words_es`
- `neg_words_es`
- `neutral_words_es`
- `normalization_es`

Consider the text "El producto provee simplicidad" (the product provides simplicity). If the word 'simplicidad' (simplicity) should be positive, it has to be loaded into the pos_words dictionary for Spanish as follows:

```
select sentimentanalysis('El producto provee simplicidad') OVER(PARTITION BEST);

 sentence | attribute | sentiment_score
-----+-----+-----
          1 | producto  |                0
          1 | simplicidad |                0
(2 rows)

insert into pulse.pos_words_es values('simplicidad');
OUTPUT
-----
          1
(1 row)

select LoadDictionary(standard USING PARAMETERS listName='pos_words') over() from pulse.pos_words_es;
Success
-----
t
(1 row)

select sentimentanalysis('El producto provee simplicidad') OVER(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | producto  |                1
          1 | simplicidad |                0
(2 rows)
```

Multilingual Examples

Language as an Argument

```
select sentimentanalysis('Cookies are sweet.', 'english') OVER(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | cookies  |                1
(1 row)
```

```
select sentimentanalysis('Las galletas son dulces', 'spanish') OVER(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | galletas |                1
(1 row)
```

The following example shows how to analyze tweets from a table where each tweet record contains the language of the tweet in addition to the text.

```

create table myTweets (text varchar(300), language varchar(15));

insert into myTweets values ('Wired reviews Amazon''s tiny-screen Kindle Fire: Web browsing sucks,
emotionally draining, makes reading a chore', 'english');

insert into myTweets values ('Cookies are sweet', 'english');

insert into myTweets values ('Why does my iPhone have 6 GB of corrupted space I can''t use? That is
obnoxious.', 'english');

insert into myTweets values ('Las galletas son dulces', 'spanish');

insert into myTweets values ('el iPhone es el celular mas popular', 'spanish');

select sentimentanalysis(text,language) OVER(PARTITION BEST) from MyTweets;

```

sentence	attribute	sentiment_score
1	reviews amazon	-1
1	kindle fire	-1
1	web	-1
1	chore	-1
1	cookies	1
1	iphone	-1
1	gb	-1
1	space	-1
1	galletas	1
1	iphone	1
1	celular	1

(11 rows)

Language as a Parameter

```

select sentimentanalysis('Las galletas son dulces' using PARAMETERS language='spanish') OVER
(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | galletas |                1
(1 row)

select sentimentanalysis('Cookies are sweet' using PARAMETERS language='english') OVER(PARTITION
BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | cookies  |                1
(1 row)

```

Although it is possible to specify the language as parameter for a specific text given in a query, using the language argument is more appropriate. The use of the language parameter is targeted to queries that analyze a set of texts (from a table) written in a same language. The language parameter is used by Pulse to skip texts in other languages because Pulse does not automatically detect the language, Thus, Pulse uses the language specified as parameter to

analyze each text from the table (consequently the sentiment scores for texts in other language may be incorrect).

The following example shows a query that analyzes tweets from a table where the tweets do not have a language value stored in the table, but are all in the same language.

```
create table myTweets (text varchar(300));

insert into myTweets values ('Las galletas son dulces');

insert into myTweets values ('el iphone es el celular mas popular');

insert into myTweets values ('el zorro rapido brinco sobre el perro flojo');

select sentimentanalysis(text using PARAMETERS language='spanish') OVER(PARTITION BEST) from
MyTweets;
```

sentence	attribute	sentiment_score
1	galletas	1
1	iphone	1
1	celular	1
1	zorro	1
1	perro	-1

(5 rows)

The following example shows a query that analyzes tweets from a table with tweets in different languages. The Spanish tweets do not have the language value. In a single query you can specify both an argument and parameter. The argument has precedence over the parameter setting. In this case the parameter is only used when a tweet doesn't provide a language value.

```
create table myTweets (doc_id int, text varchar(300), language varchar(15));

insert into myTweets values (1, 'Vertica is the best company', 'english');

insert into myTweets values (2, 'Cookies are sweet', 'english');

insert into myTweets values (3, 'The quick brown fox jumped over the lazy dog', 'english');

insert into myTweets values (4, 'Las galletas son dulces');

insert into myTweets values (5, 'el iphone es el celular mas popular');

select doc_id, sentimentanalysis(text,language using PARAMETERS language='spanish') OVER(PARTITION BY
id, text) from MyTweets;
```

doc_id	sentence	attribute	sentiment_score
1		1 vertica	1
1		1 company	1
2		1 cookies	1
3		1 fox	1
3		1 dog	-1
4		1 galletas	1

```
      5 |          1 | iphone |          1
      5 |          1 | celular |          1
(8 rows)
```

Using the Default Language

```
select sentimentanalysis('Cookies are sweet') OVER(PARTITION BEST);
 sentence | attribute | sentiment_score
-----+-----+-----
          1 | cookies   |                1
(1 row)
```

Pulse Cookbook

This section contains the following recipes for using Pulse

Batch Analyzing Data as It Is Loaded

If you are constantly loading data that needs to be analyzed with Pulse, then you should run the `sentimentAnalysis()` function in batches on the newly loaded data. You can store the sentiment scores in a separate table and associate the rows in the scored table with the original table by joining on IDs between the tables. Running `sentimentAnalysis()` as the data is loaded and storing the results is more efficient than running `sentimentAnalysis()` during interactive sessions because the `sentimentAnalysis()` can take a few seconds to return results.

For example, suppose that you are using the **Social Media Connector** (available in the *ETL and Data Ingest* section of the [Vertica Marketplace](#)) to retrieve Twitter tweets and load them into Vertica. In this case, you can create shell scripts and a cron job to automatically run `sentimentAnalysis()` on the text of the tweets. Then you can store the resulting scores in a table for quick retrieval later on.

Complete the following steps as the `dbadmin` user to run `sentimentAnalysis()` on your Twitter data. This task also sets up the system to run `sentimentAnalysis()` on new Twitter data every 2 minutes.

1. Create a table to hold the tweets (for example, named `tweets`) with the following structure:

```
create table tweets(  
  id int,  
  created_at timezonetz,  
  "user.name" varchar(144),  
  "user.screen_name" varchar(144),  
  text varchar(500),  
  "retweeted_status.retweet_count" int,  
  "retweeted_status.id" int,  
  "retweeted_status.favorite_count" int,  
  "user.location" varchar(144),  
  "coordinates.coordinates.0" float,  
  "coordinates.coordinates.1" float,  
  lang varchar(5)  
);
```

The columns are based on the data returned by Twitter's streaming API. The fields are defined in the Twitter Field Guide at <https://dev.twitter.com/docs/platform>/tweets>.

Note that the columns with quoted names; "user.name", "user.screen_name", are sub-fields within a larger field. For example, the "users" field is described here: <https://dev.twitter.com/docs/platform>/users>.

You must at least have columns for id, text, and "user.screen_name"

2. Create a table to hold the sentiment scores (for example, named : *tweet_sentiment*). Then load it with the scores from your existing tweets. Make sure no new tweets are loaded until this step completes.

Replace the column names in the following example with the column names from your twitter table. The example uses the column names used by the Social Media Connector:

```
create table tweet_sentiment as
(select id, "user.screen_name",
SentimentAnalysis(text using parameters filterlinks=true,
filterusermentions=true)
over (partition by id, "user.screen_name", text)
from tweets where lang='en' order by attribute );

-- The following table defines which data has been analyzed
create table dt_start as (select max(created_at) dt from tweets);
```

Note: If you have a large number of tweets then this command can take a long time to run. However, it is important to score your existing data before you start scoring newly loaded data.

3. Create a SQL script to update the *tweet_sentiment* table with data from newly loaded tweets. Save it in the home folder of the Vertica database admin user. For example, this path could be `/home/dbadmin/tweet_update.sql`.

Replace the column names with the column names from your twitter table. The following example uses the column names used by the Vertica Social Media Connector:

```
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
drop table if exists dt_end;
create table dt_end as (select max(created_at) dt from tweets);

-- run sentiment
insert into tweet_sentiment
(select id, "user.screen_name",
SentimentAnalysis(text using parameters filterlinks=true,
filterusermentions=true)
```

```
over (partition by id, "user.screen_name", text)
from tweets where lang='en' and
tweets.created_at > (select dt from dt_start) and
tweets.created_at <= (select dt from dt_end)
order by attribute);

-- copy date end into new start date
drop table if exists dt_start;
create table dt_start as (select dt from dt_end);

-- free up jvm resource pool memory used by this script
select release_jvm_memory();
```

4. Create a shell script named *tweet_update.sh* that is run from a cron job. This shell script runs the *tweet_update.sql* script and logs the results to the file *tweet_update.log*. Save the *tweet_update.sh* script in the home folder of the Vertica database admin user. For example, this path could be */home/dbadmin/tweet_update.sh*.

Replace the *dbadmin*, *password*, and *databasename* values with the values for your system.

```
/opt/vertica/bin/vsql -U dbadmin -w password -d databasename -f /home/dbadmin/tweet_update.sql >
tweet_update.log
```

After you have created the shell script *tweet_update.sh*, make the script executable by entering the following command: `chmod +x tweet_update.sh`.

5. Create a cron job to run the script every two minutes. Use the command `crontab -e` to create the cron job. You can view all of your created cron jobs by using the command `crontab -l`.

```
*/2 * * * * /home/dbadmin/tweet_update.sh
```

The script runs every two minutes. Any new tweets that have been loaded in that two-minute window are analyzed and the results are added to the *tweet_sentiment* table. You can join results of queries by the id's of the *tweets* and *tweet_sentiment* tables.

Analyzing Comments for a Company or Product

Pulse allows you to analyze comments (such as tweets) for a particular company or product.

For example, imagine that the fictional company *Pytell Corp* has just released a new product called *Owl-2*. You want to analyze the sentiment of both the company and the product.

You've collected several tweets from Twitter about several companies and products into your database. However, for this analysis you only want to target tweets that have to do with *Pytell Corp* and/or *Owl-2*.

The dataset for this example is below:

```
create table tweets_sample(id int, author varchar(50), text varchar(400));

insert into tweets_sample values(400900, 'DramaBugs',
'Pytell Corp has horrible customer support. On Hold 2 hours!');
insert into tweets_sample values(401200, 'Gemball',
'Owl-2 doesn't fly!');
insert into tweets_sample values(403070, 'Postta',
'Pytell finally released Owl-2!');
insert into tweets_sample values(480920, 'Instana',
'Unboxing Owl-2 after work today! Stay Tuned!');
insert into tweets_sample values(434500, 'Dailydant',
'Owl-2 flies great! I like it!');
insert into tweets_sample values(450670, 'HelpfulBen',
'Owl-2 keeps crashing into things!');
insert into tweets_sample values(402092, 'Championtips',
'Owl-2 has solved our rodent infestation!');
insert into tweets_sample values(434950, 'Editone',
'Pytell fail? Reports of Owl-2 crashing through windows. ');
insert into tweets_sample values(413956, 'CzarLatest',
'Pytell Corp's Owl-2 just released!');
insert into tweets_sample values(459988, 'CelticMiss', 'I like Ponies!');
insert into tweets_sample values(403511, 'BuffDrama',
'I am afraid of small spiders. ');
commit;
```

1. Run `SentimentAnalysis` to get an idea of how Pulse is analyzing the data:

```
SELECT author, SentimentAnalysis(text) OVER(PARTITION BY author, text) FROM tweets_sample ORDER BY attribute;
```

author	sentence	attribute	sentiment_score
-----	-----	-----	-----

DramaBugs	1	customer support	-1
Championtips	1	owl-2	0
HelpfulBen	1	owl-2	-1
Instana	1	owl-2	1
CzarLatest	1	owl-2	0
Gemball	1	owl-2	0
Postta	1	owl-2	0
Dailydant	1	owl-2	1
Editone	2	owl-2	-1
CelticMiss	1	ponies	1
Editone	2	reports	-1
Championtips	1	rodent infestation	0
BuffDrama	1	spiders	-1
Instana	2	tuned	0
Editone	1	Pytell	-1
Postta	1	Pytell	0
CzarLatest	1	Pytell corp	0
DramaBugs	1	Pytell corp	-1
Editone	2	windows	-1
Instana	1	work today	0
(20 rows)			

- There are some attributes listed (ponies!) that do not apply to the analysis that you are doing. You can focus your analysis by adding whitelist entries and filtering on the whitelist. Insert whitelist entries for the company and product name into the standard whitelist:

```
INSERT INTO pulse.white_list_en VALUES ('Pytell Corp');
INSERT INTO pulse.white_list_en VALUES ('owl-2');
commit;
```

Reload the whitelist into Pulse. Loading a user-dictionary or mapping overwrites the existing user-dictionary or mapping:

```
SELECT LoadDictionary(standard USING PARAMETERS listName='white_list') OVER() FROM pulse.white_list_en;
```

- Also, note that Pulse is not identifying all variations on the company name. There are also three obvious attributes for the product name ('Pytell', 'pytell corp). You can normalize these values by using a normalization mapping. Add the synonyms to the standard normalization mapping:

```
insert into pulse.normalization_en values('Pytell', 'Pytell Corp');
commit;
```

- Reload the normalization mapping to load the new values into Pulse:

```
SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS mapName='normalization')  
OVER() FROM pulse.normalization_en;
```

5. Run the query again to see how the normalization affects the results.

Note that 'pytell corp' has been normalized to 'pytell' and Pulse is correctly identifying the synonyms and mapping them to the base term

Determining Popular Topics

The next examples in this cookbook use a table with the following structure:

```
create table tweets(  
  id int,  
  created_at timestamptz,  
  "user.name" varchar(144),  
  "user.screen_name" varchar(144),  
  text varchar(500),  
  "retweeted_status.retweet_count" int,  
  "retweeted_status.id" int,  
  "retweeted_status.favorite_count" int,  
  "user.location" varchar(144),  
  "coordinates.coordinates.0" float,  
  "coordinates.coordinates.1" float,  
  lang varchar(5)  
);
```

The columns are based on the data returned by Twitter's streaming API. The fields are defined in the Twitter Field Guide at <https://dev.twitter.com/docs/platform/tweets>.

Note that the columns with quoted names; "user.name", "user.screen_name", are sub-fields within a larger field. For example, the "users" field is described here:

<https://dev.twitter.com/docs/platform/users>.

The example queries provided work with any Twitter data that follows the above table structure.

Determining Popular Topics

The Pulse attribute discovery feature allows you to easily find popular topics in a data set. Use the `CommentAttributes()` function to extract the attributes from rows of text and count the number of times the attribute occurs.

For example, using a dataset of 30,000 tweets that matched a keyword of "D11" collected during the [D11 tech conference](#) in 2013, you could get a count of the attributes discovered by Pulse to determine popular topics:

```
SELECT t.attribute, count(*) FROM(SELECT CommentAttributes(text)  
OVER(PARTITION BEST) FROM tweets) as t  
GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;
```

```
attribute | count  
-----+-----
```

```
http          | 3631
d11           | 3281
rt            | 2453
encryption    | 2356
usb           | 2121
aes-256       | 1859
smartphones   | 1843
rt @hp        | 1788
world         | 1609
ceo           | 1520
(10 rows)
```

If the dataset contains tweets in English and Spanish languages, then (using the Pulse multilingual version) each tweet can be analyzed according to its language by specifying the language as argument in the `CommentAttributes()` function. If the language of a specific tweet is not supported, then that tweet is ignored by the function. For example:

```
SELECT t.attribute, count(*) FROM(SELECT CommentAttributes(text,lang)
OVER(PARTITION BEST) FROM tweets) as t
GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;
```

Notice that the top attribute is "http". This is due to the large number of links in tweets. You can ignore links by using the `filterlinks` argument of `CommentAttributes()`:

```
SELECT t.attribute, count(*) FROM
(SELECT CommentAttributes(text USING PARAMETERS filterlinks=true)
OVER(PARTITION BEST) FROM tweets) as t
GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;
```

```
attribute | count
-----+-----
d11       | 4757
rt        | 2397
encryption | 2356
usb       | 2121
aes-256   | 1871
smartphones | 1829
rt @hp    | 1788
world     | 1611
ceo       | 1542
interview | 1346
(10 rows)
```

The attribute "http" is now gone from the list, but we still have "rt" (for retweet) on the list and it is not helpful in this context. You can omit terms such as "rt" by adding them to the `stop_words` list and reloading the `stop_words` user-dictionary:

```
INSERT INTO pulse.stop_words_en VALUES('rt');
commit;
SELECT LoadDictionary(standard USING PARAMETERS
listName='stop_words') OVER() FROM pulse.stop_words_en;
```

When you rerun the query you get more accurate results for the popular topics in the data set:

```
SELECT t.attribute, count(*) FROM  
(SELECT CommentAttributes(text USING PARAMETERS filterlinks=true)  
OVER(PARTITION BEST) FROM tweets) as t  
GROUP BY t.attribute  
ORDER BY count(*) DESC LIMIT 10;
```

attribute	count
d11	4757
encryption	2356
perfume	2121
usb	1871
aes-256	1829
rt @hp	1788
world	1611
ceo	1542
interview	1346
cloud	1306

(10 rows)

You can further refine the list to topics that contain specific attributes by adding the attributes in which you are interested to the `white_list`, and then filtering with the `whitelist` parameter:

```
SELECT t.attribute, count(*) FROM  
(SELECT CommentAttributes(text USING PARAMETERS filterlinks=true,  
whitelistonly=true) OVER(PARTITION BEST) FROM tweets) as t  
GROUP BY t.attribute  
ORDER BY count(*) DESC LIMIT 10;
```

Determining The Sentiment of Popular Topics

In addition to finding popular, or most discussed, topics in your data set, you can also easily get an average sentiment for the topics.

The following example uses a dataset of 10,000 tweets containing the hashtag #sports.

```
SELECT * from  
(SELECT attribute, count(attribute) AS  
cnt, AVG(sentiment_score) FROM (select  
SentimentAnalysis(text USING PARAMETERS  
filterlinks=true) OVER(PARTITION BEST) from tweets)  
AS t1 GROUP BY attribute ORDER BY  
AVG(sentiment_score) desc) AS t2  
WHERE t2.cnt > 500  
LIMIT 5;
```

The result shows the top 5 tweets with the highest average sentiment for attributes that have 500 or more occurrences:

attribute	cnt	avg
-----------	-----	-----

football		817		0.290085679314565
game		638		0.134796238244514
baseball		1558		0.128369704749679
basketball		776		0.114690721649485
hockey		2610		0.113409961685824

Determining Prolific Authors

You can identify prolific authors of your textual data without using any of the Pulse functions. For example, using the same dataset as the examples in [Determining Popular Topics](#), you can easily determine how many tweets were made by authors:

```
select "user.name", count(*) as post_count from tweets group by  
"user.name" order by count(*) DESC limit 10;
```

user.name	post_count
Nick Cicero	182
Networked Society	171
AllThingsD	137
Stephanie~	117
Jennifer Ives	105
Claudia-ElasticMinds	101
Needful Things	96
Poptart Tech	85
Patrick Bertrand	84
Alessandro Piol	81

(10 rows)

Analyzing the Sentiment of Specific Authors

You can use the `white_list` feature of `SentimentAnalysis()` to filter the attributes so only the `white_list` terms are returned. You can combine the `white_list` with a query for a list of specific authors to narrow down the results to a specific subset of authors.

Using the same `tweet_samples` table in [Analyzing Comments for a Company or Product](#), add the following sample tweets:

```
INSERT INTO tweets_sample VALUES('123', 'bcook',  
'The hyperdrive is a great machine.');
```

```
INSERT INTO tweets_sample VALUES('124', 'sprock',  
'The hyperdrive is a pinnacle of technology.');
```

```
INSERT INTO tweets_sample VALUES('125', 'tgates',  
'What is a hyperdrive?');
```

```
INSERT INTO tweets_sample VALUES('126', 'bcook', 'Roses are red.');
```

```
INSERT INTO tweets_sample VALUES('127', 'sprock',  
'Energy equals mass times the speed of light squared.');
```

```
INSERT INTO tweets_sample VALUES('128', 'tgates', 'Violets are blue.');
```

```
commit;
```

Create an `authors` table to hold the names of the authors whose sentiment you want to analyze:

```
CREATE TABLE authors (name VARCHAR, screenname VARCHAR);
```

Then insert the following authors:

```
INSERT INTO authors VALUES('Brian Cook','bcook');
```

```
INSERT INTO authors VALUES('Tom Gates', 'tgates');
```

```
INSERT INTO authors VALUES('Jim Sprock', 'sprock');
```

```
commit;
```

Add the word 'hyperdrive' to your existing `white_list` and reload the `white_list` user-dictionary:

```
INSERT INTO pulse.white_list_en VALUES('hyperdrive');
```

```
SELECT LoadDictionary(standard USING PARAMETERS  
listName='white_list') OVER() FROM pulse.white_list_en;
```

Then, you can run a query that filters on authors and the `white_list` and provides you with a sentiment score and the content of the analyzed text:

```
SELECT t1.id, t1.author, t1.attribute, t1.sentiment_score, t2.text from (SELECT id, author,  
SentimentAnalysis(text USING PARAMETERS  
whitelistonly=true) OVER (PARTITION BY id, author) FROM tweets_sample  
WHERE author IN (SELECT screenname FROM authors)) AS t1 JOIN (SELECT id,  
text FROM tweets_sample) AS t2 ON t1.id = t2.id ;
```

```
id | author | attribute | sentiment_score | text
-----+-----+-----+-----+-----
123 | bcook | hyperdrive | 1 | The hyperdrive is a great
124 | sprock | hyperdrive | 1 | The hyperdrive is a pinnacle
125 | tgates | hyperdrive | 0 | What is a hyperdrive?
(3 rows)
```

Finding Associated Attributes

Once you've analyzed your tweets and stored them in a table (see [Batch Analyzing Data as It Is Loaded](#)) you can use the analyzed data to make quick comparisons, such as finding attributes most associated with another attribute.

For example, if your primary attribute is 'microsoft', you may want to determine which other attributes are used most often with the word 'microsoft' in the same tweet. This can be accomplished with the following SQL:

```
select t1.attribute, count(*), avg(t1.sentiment_score) from tweet_sentiment t1,  
tweet_sentiment t2 where t1.id=t2.id and not t1.attribute=t2.attribute and  
t2.attribute = 'microsoft' group by t1.attribute order by count desc limit 5;
```

We get the following results from a data set of 25,000 PC Manufacturer tweets:

attribute	count	avg
windows phone	81	0.0238095238095238
power data center	77	0.58974358974359
wind project	77	0
investment	73	0
windows	57	0.175438596491228

The query allows you to gain additional insight into the scope of an attribute and may aid in determining the context of why a certain attribute it scored a certain way.

Using Pulse as an Aid in Competitive Analysis

This topic details how you can use Pulse to conduct basic competitive analysis for products or brands. Pulse makes basic competitive analysis simple through use of its white list feature. By utilizing the white list feature, you can analyze the tweets that pertain only to the brands or products that you are evaluating.

For example, say you wanted to analyze the sentiment of major food brands to determine how the brands compared to each other and what words people associate (positively and negatively) about the brands. Your work flow to do this analysis with Twitter and Vertica Pulse could be as follows:

1. Start collecting tweets based on the brands or products that you are following. For example, you can use the Social Media Connector (available on the Pulse [marketplace](#)) to collect tweets matching keywords.
2. First, create a `white_list` that contains the same keywords as the tweets that you are collecting. The whitelist allows you to later group and filter tweets collected. For example:

```
insert into pulse.white_list_en values ('productA');
insert into pulse.white_list_en values ('productB');
insert into pulse.white_list_en values ('productC');
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
```

3. [Batch Load Tweets](#), and be sure to specify `whitelistonly=true` and `relatedwords=true` in the `SentimentAnalysis()` function. This creates a table with the sentiment score for your white-listed attributes. Note that this should be done in batches for large data sets. For smaller data sets (depending on your hardware) you can try and analyze all the tweets at once. For example:

```
create table tweet_sentiment as
(select id, "user.screen_name",
SentimentAnalysis(text using parameters filterlinks=true,
filterusermentions=true, relatedwords=true,
filterretweets=true, whitelistingonly=true)
over (partition by id, "user.screen_name", text)
from tweets where lang='en' order by attribute );
```

4. Verify that your `tweet_sentiment` table contains only your whitelist attributes. The following query should only return the brands/products that you have white listed. For example:

```
=> select distinct(attribute) from tweet_sentiment;

 attribute
-----
ProductA
ProductB
ProductC
(3 rows)
```

5. You can get a basic idea of which product or brand is being talked about the most by seeing how many instances of each attribute appear in your data set:

```
=> select attribute, count(*) from tweet_sentiment group by (attribute) order by count(*) desc;
 attribute | count
-----+-----
ProductA  |    701
ProductB  |    192
ProductC  |     52
(3 rows)
```

You can see that ProductA is the most talked about product of three being analyzed over the time-frame that the tweets were collected.

6. Determine the average sentiment scores of the tweets you have collected:

```
=> select attribute, avg(sentiment_score) as score from tweet_sentiment group by (attribute)
order by score DESC;

 attribute |          score
-----+-----
ProductC  |  0.192307692307692
ProductB  | -0.0729166666666667
ProductA  | -0.122681883024251
(3 rows)
```

From this basic analysis, you can see that ProductC has the most positive sentiment from the three brands being analyzed over the time period when the tweets were collected, and ProductA has the lowest sentiment.

7. You can also determine which words or phrases are associated with each attribute in their positive and negative contexts. For example, to see the list of words that are most

associated with positive sentiment for ProductC, you can look at the related words fields and add up the occurrences of words associated with positive sentiment:

```
=> select count(*), related_word_1 from tweet_sentiment where attribute = 'ProductC' and sentiment_score > 0 group by related_word_1 order by count DESC;
```

```
count | related_word_1
-----+-----
    11 | delicious
     2 | love
     1 | best
     1 | bless
     1 | good
     1 | work
(6 rows)
```

You can also do the same for negative sentiment:

```
=> select count(*), related_word_1 from tweet_sentiment where attribute = 'ProductC' and sentiment_score < 0 group by related_word_1 order by count DESC;
```

```
count | related_word_1
-----+-----
     1 | working
     1 | dragging
     1 | bad
     1 | doomed
     1 | loud
     1 | stressful
     1 | damn
(7 rows)
```

8. Finally, Pulse makes it easy to see other attributes associated with your target attributes to help you better understand the context in which people are discussing the brands or products that you are analyzing.

a. Create another sentiment table from your data, but this time omit the `whitelistonly` and `relatedwords` parameters:

```
create table tweet2_sentiment as
(select id, "user.screen_name",
SentimentAnalysis(text using parameters filterlinks=true,
filterusermentions=true, filterretweets=true)
over (partition by id, "user.screen_name", text)
from tweets where lang='en' order by attribute );
```

b. Next, query the tweets that contain your target attribute and find all the other attributes associated with those tweets. Display a count of the top 5 attributes (not

including the target attribute):

```
=> select count(attribute), attribute from tweet2_sentiment where id in (select id from
tweet_sentiment where attribute = 'ProductC') and attribute <> 'ProductC' group by
(attribute) order by count(attribute) DESC limit 5;
count | attribute
-----+-----
    13 | bbq
    11 | state
    11 | sandwich
    11 | steak
     3 | ProductB
(5 rows)
```

As you can see, a few basic queries can tell you the general sentiment differences between multiple brands or products. You can also determine which words are contributing to the sentiment of each product/brand that you are analyzing and which other attributes people are talking about when they mention the brand or product(s) that you are analyzing.

You could further refine these queries by breaking out different geographic locations or time of day by joining the IDs of the tweet_sentiment table back to the main tweets table and filtering by location or time.

Pulse Function Reference

LoadDictionary

Lists words from a Pulse user-defined dictionary into memory for use by sentimentAnalysis() and other Pulse functions.

This function must be used with the OVER() clause.

For more information on Pulse user-defined dictionaries, see [Dictionaries and Mappings](#).

Syntax

```
SELECT LoadDictionary(word USING PARAMETERS listName='Listname' [, language='Lang'] [, label='Label'])  
OVER() FROM table;
```

Parameters

Argument	Description
<i>word</i>	A column of words to assign to a user-dictionary list. The column name must match the value of word.
listName	The user-dictionary list from which to load the values from <i>word</i> . Valid values: <ul style="list-style-type: none">• pos_words• neg_words• neutral_words• stop_words• white_list See Dictionaries and Mappings for details on each list type.
language	The language of the dictionary: <ul style="list-style-type: none">• 'english' or 'en'

Argument	Description
	<ul style="list-style-type: none">'spanish' or 'es'
label	The label that you want to assign to the dictionary.
table	The specified table from which values are loaded.

Examples

```
SELECT LoadDictionary(standard USING PARAMETERS listName=
'neg_words_en') OVER() from pulse.neg_words_en;

SELECT LoadDictionary(standard USING PARAMETERS listName=
'pos_words_en') OVER() from pulse.pos_words_en;

SELECT LoadDictionary(standard USING PARAMETERS listName=
'pos_words_en', language='english') OVER() from pulse.pos_words_en;

SELECT LoadDictionary(standard USING PARAMETERS listName=
'pos_words_es', language='spanish') OVER() from pulse.pos_words_es;

SELECT LoadDictionary(standard USING PARAMETERS listName=
'neg_words',label='custom_negatives') OVER() from pulse.neg_words_en;
```

See Also

- [LoadMapping\(\)](#)
- [GetLoadedDictionary\(\)](#)
- [GetStorage\(\)](#)

LoadMapping

Loads a Pulse user-mapping into memory for use by `sentimentAnalysis()` and other Pulse functions.

This function must be used with the `OVER()` clause.

For more information on Pulse user-mappings, see [Dictionaries and Mappings](#).

Syntax

```
SELECT LoadMapping(base, wordToMap USING PARAMETERS mapName='mapName' [, language='Lang'][,  
label='label']) OVER() FROM table;
```

Parameters

Argument	Description
<code>base</code>	A column of base words to assign to a mapped word. The column name must match the value of <code>base</code> .
<code>wordToMap</code>	A column of words to map to the base word in the same row. The column name must match the value of <code>wordToMap</code> .
<code>mapName</code>	The mapping to load the words into. Valid values: <ul style="list-style-type: none"><code>irregular_verbs</code> — list of conjugations of verbs and their bases.<code>normalization</code> — list of synonyms and their base word.
<code>language</code>	The language of the dictionary: <ul style="list-style-type: none"><code>'english'</code> or <code>'en'</code><code>'spanish'</code> or <code>'es'</code>
<code>label</code>	The label of the mapping that you want to load. If you do not provide a label, Pulse uses the default mapping.
<code>table</code>	The specified table from which values are loaded.

Examples

```
SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS
mapName='normalization') OVER() from pulse.normalization_en;

SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS
mapName='normalization', language='english') OVER() from pulse.normalization_en;

SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS
mapName='normalization', language='spanish') OVER() from pulse.normalization_es;
```

See Also

- [LoadDictionary\(\)](#)
- [GetLoadedMapping\(\)](#)
- [GetStorage\(\)](#)

SentimentAnalysis

Provides a sentiment score for each attribute (noun) in a given body of text. Positive sentiment receives a positive integer score and negative sentiment receives a negative integer score. A score of 0 indicates that the sentiment for the attribute is neutral.

This function must be used with the `OVER()` clause. Use `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `OVER()` clause. Any valid `PARTITION BY` clause is acceptable. However, only the `PARTITION BY` clause which matches the segmentation clause of the table's projection provides optimum performance. You can improve performance by segmenting on the columns in the `PARTITION BY` clause.

Syntax

```
SentimentAnalysis(text [, 'Language'] [ USING PARAMETERS  
[ whitelistonly = boolean ]  
[, filterlinks = boolean ]  
[, filterusermentions = boolean ]  
[, filterhashtags = boolean ]  
[, filterpunctuation = boolean ]  
[, filterretweets = boolean ]  
[, relatedwords = boolean ]  
[, adjustcasing = boolean ]  
[, language = string ]  
[, label='Label']  
[, granularity='ASD']  
[, actionPattern='boolean']  
)
```

Note: language can be specified as an argument and/or as a parameter. When specified as both, the argument value supersedes the parameter value.

Parameters

Argument	Description
text	The text to analyze. Limited to 65,000 bytes.
whitelistonly	Optional. Default false. When set to true only attributes defined in the whitelist user-dictionary are scored. Use this setting to limit your analysis to the objects of action patterns .

Argument	Description
filterlinks	Optional. Default false. When set to true, links are not included as attributes.
filterusermentions	Optional. Default false. When set to true, Twitter user mentions (@username) are not included as attributes.
filterhashtags	Optional. Default false. When set to true, Twitter hashtags (#hashtag) are not included as attributes.
filterpunctuation	Optional. Default true. Filters any punctuation that occurs at the beginning of an attribute other than @ and #.
filterretweets	Optional. Defaults to false. Filters out the characters "RT" from re-tweets in attributes.
relatedwords	Optional. Defaults to false. When set to true, provides up to three words from the sentence used to help determine the sentiment of the attribute.
adjustcasing	Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries.
language	The language: <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
label	Optional. The label of the dictionaries that you want to use for sentiment analysis. If you do not include a label, Pulse uses the default dictionaries.
granularity	Optional. The level of the sentiment analysis that you want to perform: <ul style="list-style-type: none">• A — Attribute level analysis• S — Sentence level analysis

Argument	Description
	<ul style="list-style-type: none"> D — Document level analysis <p>You can specify any granularity level or combination of levels with your sentiment analysis. If you do not specify a granularity level, Pulse performs an attribute level analysis.</p>
actionPattern	Optional. Default false. When set to true checks for action patterns in the analyzed content.

Examples

These examples show various ways you can use Pulse to detect user sentiment.

Query for sentiment in the following sentence.

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.') OVER(PARTITION BEST);
```

```

sentence | attribute | sentiment score
-----+-----+-----
      1 | fox      |           1
      1 | dog      |          -1
(2 rows)

```

Query to identify the words that triggered the sentiment score.

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.'
USING PARAMETERS relatedwords=true) OVER(PARTITION BEST);
```

```

sentence | attribute | sentiment_score | related_word_1 | related_word_2 | related_word_3
-----+-----+-----+-----+-----+-----
      1 | fox      |           1 | quick          | lazy           |
      1 | dog      |          -1 | lazy           |                |
(2 rows)

```

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.', 'english')
OVER(PARTITION BEST);
```

```

sentence | attribute | sentiment_score
-----+-----+-----
      1 | fox      |           1
      1 | dog      |          -1
(2 rows)

```

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.'
using PARAMETERS language='english') OVER(PARTITION BEST);
```

```

sentence | attribute | sentiment_score
-----+-----+-----
      1 | fox      |           1
      1 | dog      |          -1

```

(2 rows)

```
SELECT SentimentAnalysis('El zorro rapido brinco sobre el perro flojo.',  
'spanish') OVER(PARTITION BEST);
```

sentence	attribute	sentiment_score
1	zorro	1
1	perro	-1

(2 rows)

```
SELECT SentimentAnalysis('El zorro rapido brinco sobre el perro flojo.'  
using PARAMETERS language='spanish') OVER(PARTITION BEST);
```

sentence	attribute	sentiment_score
1	zorro	1
1	perro	-1

(2 rows)

```
SELECT SentimentAnalysis('The camera takes great quality pictures but is  
expensive. It feels like a professional one.'  
USING PARAMETERS granularity='ASD') over();
```

sentence	attribute	sentiment_score	mixed
		1	true
1		0	true
2		1	false
1	camera	1	
1	quality pictures	1	

```
SELECT sentimentAnalysis('Right after school on November 8th I will go to target, walmart, and best  
buy and buy #blueslidepark just for @MacMiller' USING PARAMETERS  
actionPattern=true,whitelistonly=true) over();
```

sentence	attribute	sentiment_score	action	action_pattern
1	walmart	1	go to target	#action{\$verb \$prep \$verb}
1	walmart	1	go to target	#action{\$verb to \$verb}

(2 rows)

Getting Twitter User-Mentioned Sentiment

```
SELECT SentimentAnalysis('@company is great!') OVER(PARTITION BEST);
```

sentence	attribute	sentiment score
1	@company	1

(1 row)

Filtering Twitter User Sentiment

```
SELECT SentimentAnalysis('@company is great!' USING PARAMETERS
filterusermentions=true) OVER(PARTITION BEST);
 sentence | attribute | sentiment score
-----+-----+-----
(0 rows)
```

See Also

- [LoadDictionary\(\)](#)
- [LoadMapping\(\)](#)
- [ExtractSentence\(\)](#)
- [GetSentenceCount\(\)](#)
- [GetAllSentences\(\)](#)
- [CommentAttributes\(\)](#)

PartsOfSpeech

Tags the words in one or more sentences with their part of speech classification, using Penn Treebank parts of speech tags.

Syntax

```
SELECT PartsOfSpeech('sentences'[, language='Lang'] [using PARAMETERS [ language='Lang'] [, adjustcasing=boolean]) OVER(PARTITION BEST);
```

Parameters

Argument	Description
<i>sentences</i>	One or more sentences to be tagged with parts of speech markup.
language	The language: <ul style="list-style-type: none">'<i>english</i>' or '<i>en</i>''<i>spanish</i>' or '<i>es</i>'
adjustcasing	Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries.

Notes

- This function returns a part of speech markup for each word. The markup used is the [Penn Treebank Project Parts of Speech Tags](#) while for Spanish the [Parole Reduced Tagset](#) is used.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.

Examples

```
select partsOfSpeech('The quick brown fox jumped over the lazy dog.') OVER(PARTITION BEST);
sentence | token | part_of_speech
-----+-----+-----
1 | the | DT
1 | quick | JJ
1 | brown | JJ
1 | fox | NN
1 | jumped | VBD
1 | over | IN
1 | the | DT
1 | lazy | JJ
1 | dog | NN
1 | . | .
(10 rows)
```

```
select partsOfSpeech('Every good boy deserves fudge.') OVER(PARTITION BEST);
sentence | token | part_of_speech
-----+-----+-----
1 | every | DT
1 | good | JJ
1 | boy | NN
1 | deserves | VBZ
1 | fudge | NN
1 | . | .
(6 rows)
```

```
select partsOfSpeech('The quick brown fox jumped over the lazy dog.', 'english')
OVER(PARTITION BEST);
```

```
sentence | token | part_of_speech
-----+-----+-----
1 | the | DT
1 | quick | JJ
1 | brown | JJ
1 | fox | NN
1 | jumped | VBD
1 | over | IN
1 | the | DT
1 | lazy | JJ
1 | dog | NN
1 | . | .
(10 rows)
```

```
select partsOfSpeech('El zorro rapido brinco sobre el perro flojo','spanish')
over();
sentence | token | part_of_speech
-----+-----+-----
1 | El | DA
1 | zorro | NC
1 | rapido | AQ
1 | brinco | AQ
```

```
1 | sobre | SP
1 | el | DA
1 | perro | NC
1 | flojo | AQ
(8 rows)
```

See Also

- [SentimentAnalysis\(\)](#)

GetAllDictionarySetLabels

Lists all the dictionary labels that are loaded into the current Pulse session. This function shows you which labels are currently in use. You can load only one dictionary of each type in a single session.

Syntax

```
SELECT GetAllDictionarySetLabels() OVER();
```

Examples

```
SELECT GetAllDictionarySetLables() OVER();
  label
-----
default
sports_teams
(2 rows)
```

GetAllDictionaryWords

Lists all dictionary words that are currently loaded into Pulse. This function can help you determine which user-defined words in a sentence might be affecting the sentiment score of an attribute.

Syntax

```
SELECT GetAllDictionaryWords([using PARAMETERS language='Language'[, label='Label']] OVER());
```

Parameters

Argument	Description
language	The language of the dictionary: <ul style="list-style-type: none">'english' or 'en''spanish' or 'es'
label	The label of the dictionaries that you want to list. If you do not provide a label, Pulse uses the default dictionaries.

Examples

```
SELECT GetAllDictionaryWords() OVER();
dictionary | word
-----+-----
neg_words  | ratchet
neg_words  | squirelly

select GetAllDictionaryWords(using parameters language='english') over();
dictionary | word
-----+-----
pos_words_en | simplicity
(1 row)

select GetAllDictionaryWords(using parameters label='music') over();
dictionary | word
```

```
-----+-----  
white_list_en | classical  
white_list_en | popular  
white_list_en | rock  
(3 rows)
```

See Also

- [GetAllMappingWords\(\)](#)

GetAllLoadedDictionaries

Lists all the dictionaries and dictionary labels that are loaded into the current Pulse session. This function shows you which dictionaries are determining the sentiment score of an attribute. Only one dictionary of each type can be loaded in a single session.

Syntax

```
SELECT GetAllLoadedDictionaries() OVER();
```

Examples

```
SELECT GetAllLoadedDictionaries() OVER();
```

dictionary	label
neg_words_en	default
stop_words_es	default
neutral_words_es	default
white_list_en	default
normalization_en	default
pos_words_es	default
neg_words_es	default
pos_words_en	default
white_list_es	default
neutral_words_en	default
stop_words_en	default
normalization_es	default

(12 rows)

GetAllMappingWords

Lists all user-defined bases and synonyms that are currently loaded into Pulse. This function helps you determine which user-defined mappings in a sentence might be affecting the sentiment score of an attribute.

Syntax

```
SELECT GetAllMappingWords([using PARAMETERS language='Language' ][, label='Label']) OVER();
```

Parameters

Argument	Description
language	The language of the dictionary: <ul style="list-style-type: none">'english' or 'en''spanish' or 'es'
label	The label of the mappings that you want to list. If you do not provide a lable, Pulse uses the default dictionaries.

Examples

```
SELECT GetAllMappingWords() OVER() limit 10;
 mapping | key | value
-----+-----+-----
normalization | hp | hewlett packard
normalization | hp | hewlett-packard
normalization | companycorp | company-corp
normalization | companycorp | companycorps
normalization | companycorp | companycorp's
normalization | producthd | product hd
normalization | producthd | product-hd
normalization | companycorp | company corp
(8 rows)
```

```
select getAllMappingWords(using parameters language='english') over();
      mapping      | key |      value
-----+-----+-----
normalization_en  | hp  | hewlett-packard
normalization_en  | hp  | hewlett Packard
(2 rows)

select getAllMappingWords(using parameters language='spanish') over();
      mapping      | key |      value
-----+-----+-----
normalization_es  | hidalgo | miguel hidalgo
(1 row)
```

See Also

- [GetAllDictionaryWords\(\)](#)

CommentAttributes

Retrieves the attributes (nouns) from a given piece of text.

Syntax

```
CommentAttributes(text[,Language][ USING PARAMETERS  
[ whitelistonly = boolean ]  
[, filterlinks = boolean ]  
[, filterusermentions = boolean ]  
[, filterhashtags = boolean ]  
[, filterpunctuation = boolean ]  
[, filterretweets = boolean ]  
[, adjustcasing = boolean ]  
[, language = string ]])  
])
```

Parameters

Argument	Description
text	The text from which to extract the attributes.
language	The language: <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
whitelistonly	Optional. Default false. When set to true only attributes defined in the white_list user-dictionary are returned.
filterlinks	Optional. Default false. When set to true, links are not set as attributes.
filterusermentions	Optional. Default false. When set to true, Twitter usernames (@username) are not set as attributes.
filterhashtags	Optional. Default false. When set to true, removes the following from tweets: <ul style="list-style-type: none">• hashtag symbols - For example, #pizza becomes pizza.

Argument	Description
	<ul style="list-style-type: none">• @mentions - For example, Vertica would remove @NewYorkCity from a tweet.• Link URLs
filterpunctuation	Optional. Default true. Filters any punctuation that occurs at the beginning of an attribute other than @ and #.
filterretweets	Optional. Defaults to false. Filters out the characters "RT" from re-tweets in attributes.
adjustcasing	Optional. Defaults to false. When set to true, all letters in the sentence are converted to upper-case before sentence detection. After sentence detection all letters are converted to lower-case. This option is helpful if the original data is all in lower-case and Pulse is incorrectly identifying parts of speech in the sentence.

Notes

- The text argument is limited to 65,000 bytes.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
- *language* can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

Examples

```
select CommentAttributes('The quick brown fox jumped over the lazy dog. All good boys deserve fudge.') OVER(PARTITION BEST);
sentence | attribute
-----+-----
1 | fox
1 | dog
2 | boys
2 | fudge
(4 rows)
```

```
select commentattributes('the quick brown fox jumped over the lazy dog. All good boys deserve fudge', 'english') over();
```

```
sentence | attribute
-----+-----
      1 | fox
      1 | dog
      2 | boys
      2 | fudge
(4 rows)
```

```
select commentattributes('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'
using parameters language='english') over();
```

```
sentence | attribute
-----+-----
      1 | fox
      1 | dog
      2 | boys
      2 | fudge
```

```
select commentattributes('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos
merecen un premio'
```

```
, 'spanish') over();
sentence | attribute
-----+-----
      1 | zorro
      1 | perro
      2 | chicos
      2 | premio
(4 rows)
```

```
select commentattributes('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos
merecen un premio'
```

```
using PARAMETERS language='spanish') over();
```

```
sentence | attribute
-----+-----
      1 | zorro
      1 | perro
      2 | chicos
      2 | premio
(4 rows)
```

Filtering User-mentions

```
SELECT CommentAttributes('@user is always late. He kept me waiting 20 minutes last weekend.'
USING PARAMETERS filterusermentions=true) OVER(PARTITION BEST);
```

```
sentence | attribute
-----+-----
      2 | weekend
(1 row)
```

See Also

- [SentimentAnalysis\(\)](#)

GetSentenceCount

Returns the number of sentences in a body of text. You can use this function to count the number of sentences in a long piece of text. It is also useful if you are programmatically using the [ExtractSentence](#) function and need to know the number of sentences in a piece of text.

Syntax

```
select GetSentenceCount(text [, Language] [ USING PARAMETERS
[ filterlinks = boolean ]
[, filterusermentions = boolean ]
[, filterhashtags = boolean ]
[, adjustcasing = boolean ]
[, language = string ]
])
```

Parameters

Argument	Description
text	The text from which to extract the number of sentences. Currently English and Spanish language text are supported for analysis.
language	The language: <ul style="list-style-type: none">'<i>english</i>' or '<i>en</i>''<i>spanish</i>' or '<i>es</i>'
filterlinks	Optional. Default false. When set to true, sentences that are only links are not counted as a sentence.
filterusermentions	Optional. Default false. When set to true, sentences that are only Twitter user mentions (@username) are not counted as a sentence.
filterhashtags	Optional. Default false. When set to true, sentences that are only Twitter hashtags (#hashtag) are not counted as a sentence.
adjustcasing	Optional. Defaults to false. When set to true, all letters in the sentence are converted to upper-case before sentence detection.

Argument	Description
	After sentence detection all letters are converted to lower-case. This option is helpful if the original data is all in lower-case and Pulse is incorrectly identifying parts of speech in the sentence.

Notes

- The text argument is limited to 65,000 bytes.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
- *language* can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

Examples

```
SELECT GetSentenceCount('The quick brown fox jumped over the lazy dog. Every good boy deserves fudge') OVER(PARTITION BEST);
```

```
  sentence_count
```

```
-----
```

```
2
```

```
(1 row)
```

```
SELECT getsentencecount('http://hp.com. @hp. http://hp.com is great!') OVER(PARTITION BEST);
```

```
  sentence_count
```

```
-----
```

```
3
```

```
(1 row)
```

```
select getsentencecount('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio'
```

```
  using PARAMETERS language='spanish') over();
```

```
  sentence_count
```

```
-----
```

```
2
```

```
(1 row)
```

```
select getsentencecount('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio'
```

```
  , 'spanish') over();
```

```
  sentence_count
```

```
-----
```

```
2
```

```
(1 row)
```

```
select getsentencecount('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'
```

```
using parameters language='english') over();
sentence_count
-----
                2
(1 row)

select getsentencecount('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'
, 'english') over();
sentence_count
-----
                2
(1 row)
```

Filtering Links and User Mentions

```
SELECT GetSentenceCount('http://hp.com. @hp. http://hp.com is great!' USING PARAMETERS
filterlinks=true, filterusermentions=true) OVER(PARTITION BEST);
sentence_count
-----
                1
(1 row)
```

See Also

- [GetAllSentences\(\)](#)
- [ExtractSentence\(\)](#)

ExtractSentence

Returns the specified sentence from a body of text.

Syntax

```
ExtractSentence(text, sentence [, Language] [USING PARAMETERS  
[ filterlinks = boolean ]  
[, filterusermentions = boolean ]  
[, filterhashtags = boolean ]  
[, adjustcasing = boolean ]  
[, language = string ]  
)
```

Parameters

Argument	Description
text	The text containing the sentence to extract.
language	The language: <ul style="list-style-type: none">• '<i>english</i>' or '<i>en</i>'• '<i>spanish</i>' or '<i>es</i>'
sentence	Integer value. The number of the sentence in the text .
filterlinks	Optional. Default false. When set to true, sentences that are only links are skipped over and ignored. Any links in a sentence are not included in the extracted sentence.
filterusermentions	Optional. Default false. When set to true, sentences that are only Twitter user mentions (@username) are skipped over and ignored. Any user-mentions in a sentence are not included in the extracted sentence.
filterhashtags	Optional. Default false. When set to true, sentences that are only Twitter hashtags (#hashtag) are skipped over and ignored. Any hashtags in a sentence are not included in the extracted sentence.

Argument	Description
adjustcasing	Optional. Defaults to false. When set to true, all letters in the sentence are converted to upper-case before sentence detection. After sentence detection all letters are converted to lower-case. This option is helpful if the original data is all in lower-case and Pulse is incorrectly identifying parts of speech in the sentence.

Notes

- The text argument is limited to 65,000 bytes.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
- *language* can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

Examples

```
select ExtractSentence('The quick brown fox jumped. Every good boy deserves fudge', 2)
OVER(PARTITION BEST);
      sentence
-----
Every good boy deserves fudge.
(1 row)
select extractSentence('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'
, 2, 'english') over();
      sentence
-----
All good boys deserve fudge
(1 row)
select extractSentence('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'
,2 using parameters language='english') over();
      sentence
-----
All good boys deserve fudge
(1 row)
select extractSentence('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen
un premio'
, 2, 'spanish') over();
      sentence
-----
Todos los chicos buenos merecen un premio
(1 row)
```

```
select extractSentence('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio'
,2 using parameters language='spanish') over();
      sentence
-----
Todos los chicos buenos merecen un premio
(1 row)
```

Filtering Links

```
SELECT ExtractSentence('HP - http://hp.com is a useful website. I
like HP.', 1 USING PARAMETERS filterlinks=true) OVER(PARTITION BEST);
      sentence
-----
hp - is a useful website.
(1 row)
```

See Also

- [GetSentenceCount\(\)](#)
- [GetAllSentences\(\)](#)

GetAllSentences

Extracts a row for each sentence in a body of text. This ability is useful if you need to programmatically get each sentence in a piece of text.

Syntax

```
GetAllSentences(text [, language[ USING PARAMETERS  
[ filterlinks = boolean ]  
[, filterusermentions = boolean ]  
[, filterhashtags = boolean ]  
[, adjustcasing = boolean ]  
[, language = string ]  
])
```

Parameters

Argument	Description
text	The text from which to get the sentences.
language	The language: <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
filterlinks	Optional. Default false. When set to true, sentences that are only links are skipped over and ignored. Any links in a sentence are not included in the extracted sentence.
filterusermentions	Optional. Default false. When set to true, sentences that are only Twitter user mentions (@username) are skipped over and ignored. Any user-mentions in a sentence are not included in the extracted sentence.
filterhashtags	Optional. Default false. When set to true, sentences that are only Twitter hashtags (#hashtag) are skipped over and ignored. Any hashtags in a sentence are not included in the extracted sentence.

Argument	Description
adjustcasing	Optional. Defaults to false. When set to true, all letters in the sentence are converted to upper-case before sentence detection. After sentence detection all letters are converted to lower-case. This option is helpful if the original data is all in lower-case and Pulse is incorrectly identifying parts of speech in the sentence.

Notes

- The text argument is limited to 65,000 bytes.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
- *language* can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

Examples

```
SELECT GetAllSentences('The quick brown fox jumped over the lazy  
dog. Every good boy deserves fudge') OVER(PARTITION BEST);
```

```
          sentence  
-----  
The quick brown fox jumped over the lazy dog.  
Every good boy deserves fudge.  
(2 rows)
```

```
select getAllSentences('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'  
, 'english') over();  
sentence_index |          sentence_text  
-----+-----  
1 | the quick brown fox jumped over the lazy dog.  
2 | All good boys deserve fudge  
(2 rows)
```

```
select getAllSentences('the quick brown fox jumped over the lazy dog. All good boys deserve fudge'  
using parameters language='english') over();  
sentence_index |          sentence_text  
-----+-----  
1 | the quick brown fox jumped over the lazy dog.  
2 | All good boys deserve fudge  
(2 rows)
```

```
select getAllSentences('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio'
, 'spanish') over();
sentence_index | sentence_text
-----+-----
              1 | el zorro rapido brinco sobre el perro flojo.
              2 | Todos los chicos buenos merecen un premio
(2 rows)

select getAllSentences('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio'
using parameters language='spanish') over();
sentence_index | sentence_text
-----+-----
              1 | el zorro rapido brinco sobre el perro flojo.
              2 | Todos los chicos buenos merecen un premio
(2 rows)
```

Filtering User-mentions

```
SELECT GetAllSentences('@user is always late. He kept me waiting 20 minutes last time.'
USING PARAMETERS filterusermentions=true)
OVER(PARTITION BEST);
sentence
-----
is always late.
he kept me waiting 20 minutes last time.
(2 rows)
```

See Also

- [GetSentenceCount\(\)](#)
- [ExtractSentence\(\)](#)

SetDefaultLanguage

Sets the new default language to use for Pulse functions if no language is specified in a Pulse function call.

Syntax

`SetDefaultLanguage(Language)`

Parameters

Argument	Description
language	The language: <ul style="list-style-type: none">'<i>english</i>' or '<i>en</i>''<i>spanish</i>' or '<i>es</i>'

Notes

- This function must be used with the OVER() clause.
- The default language immediately after installation is English.
- The language that is set when using this function is the default language across all sessions and is persistent across database restarts.

Examples

```
=> select setDefaultLanguage('es') over();
Success
-----
t
(1 row)
```

See Also

- [SentimentAnalysis](#)

GetLoadedDictionary

Lists the currently loaded words for the specified user-dictionary.

If the user-dictionary is not loaded, then nothing is returned. You must use the OVER() clause with this function.

Syntax

```
SELECT GetLoadedDictionary(user-dictionary [using PARAMETERS language = string][, label='Label']) OVER  
( );
```

Parameters

Argument	Description
user-dictionary	<p>The user-dictionary list to retrieve.</p> <p>Valid values:</p> <ul style="list-style-type: none">• pos_words• neg_words• neutral_words• stop_words• white_list <p>See Dictionaries and Mappings for details on each type.</p>
language	<p>The language of the dictionary:</p> <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
label	<p>The label of the dictionaries that you want to list. If you do not provide a label, Pulse uses the default dictionaries.</p>

Examples

Note: This example is from a three node cluster, so three copies of the words are returned.

```
SELECT GetLoadedDictionary('pos_words') OVER();

      word
-----
:-)
adequate
admire
admiringly
adore
adoringly
adulation
adventuresome
advocated
affable
affably
affordable
affordably
afordable
all-around
alluringly
amazement
ameliorate
ample
amusing
--More--

SELECT GetLoadedDictionary('pos_words' using PARAMETERS language='english') OVER();
      word
-----
simplicity
(1 row)

SELECT GetLoadedDictionary('pos_words' using PARAMETERS language='spanish') OVER();
      word
-----
simplicidad
(1 row)
```

See Also

- [LoadDictionary\(\)](#)
- [GetLoadedMapping\(\)](#)

GetLoadedMapping

Lists the currently loaded words for the specified user-defined mapping.

If the mapping is not loaded with [LoadMapping](#), then nothing is returned. This function must be used with the OVER() clause.

Syntax

```
SELECT GetLoadedMapping('normalization' [using PARAMETERS language = string][, label='Label']) OVER();
```

Parameters

Argument	Description
mapping	The mapping list to retrieve. Currently the only mapping supported is: normalization Note: By default, the normalization list is empty.
language	The language of the dictionary: <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
label	The label to which you want to load the specified mapping. If you do not include a label, Pulse loads the default UDDs.

Examples

```
SELECT GetLoadedMapping('normalization') OVER();
key | value
-----+-----
hp | hewlett packard
(1 row)
```

```
SELECT GetLoadedMapping('normalization' using PARAMETERS language='english') OVER();
key |      value
-----+-----
hp  | hewlett-packard
hp  | hewlett packard
(2 rows)

SELECT GetLoadedMapping('normalization' using PARAMETERS language='spanish') OVER();
key |      value
-----+-----
hidalgo | miguel hidalgo
(1 row)
```

See Also

- [LoadMapping\(\)](#)
- [GetLoadedDictionary\(\)](#)

GetStorage

Lists the currently loaded user-dictionaries and user-defined mapping.

This function must be used with the OVER() clause.

Syntax

```
SELECT GetStorage([using PARAMETERS label='Label']) OVER();
```

Parameters

Argument	Description
label	The label of the dictionaries and mapping names that you want to list. If you do not provide a label, Pulse uses the default dictionaries.

Examples

```
SELECT GetStorage() OVER();
```

key
neg_words_en
neutral_words_en
pos_words_en
stop_words_en
white_list_en
normalization_en
neg_words_es
neutral_words_es
pos_words_es
stop_words_es
white_list_es
normalization_es

(12 rows)

See Also

- [LoadDictionary\(\)](#)
- [LoadMapping\(\)](#)
- [GetLoadedDictionary\(\)](#)
- [GetLoadedMapping\(\)](#)

UnloadLabeledDictionary

Unloads a specific dictionary from a Pulse session. The dictionary continues to exist, and a user can later reload the dictionary, if needed.

You cannot unload a default dictionary, but you can replace it by loading a custom user-defined dictionary.

Syntax

```
SELECT unloadLabeledDictionary(USING PARAMETERS listname='Listname'[, language='Lang'] [, label='Label'])  
OVER();
```

Parameters

Argument	Description
listName	The type of the dictionary that you want to unload. listName must be one of: <ul style="list-style-type: none">• pos_words• neg_words• neutral_words• stop_words• white_list See Dictionaries and Mappings for details on each list type.
language	The language: <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
label	The label of the dictionary that you want to unload.

Examples

```
select unloadLabeledDictionary(USING PARAMETERS listname='neg_words',  
label='custom_negatives') OVER();
```

```
success  
-----  
t  
(1 row)
```

See Also

- [UnloadLabeledDictionarySet\(\)](#)

UnloadLabeledDictionarySet

Unloads all user-defined dictionaries with a particular label from a Pulse session. The dictionaries continue to exist, and a user can later reload the dictionaries, if needed.

You cannot unload a default dictionary, but you can replace it by loading a custom user-defined dictionary.

Syntax

```
SELECT unloadLabeledDictionarySet(USING PARAMETERS label='LabelName') OVER();
```

Parameters

Argument	Description
<i>label</i>	The label of the dictionary set that you want to unload.

Examples

```
select unloadLabeledDictionarySet(USING PARAMETERS label='custom_negatives') OVER();

success
-----
t
(1 row)
```

See Also

- [UnloadLabeledDictionary\(\)](#)

UnloadLabeledMapping

Unloads a specific mapping from a Pulse session. The mapping continues to exist, and a user can later reload it, if needed.

Syntax

```
SELECT unloadLabeledMapping(USING PARAMETERS mapName='normalization' [, language='Lang'][,  
label='Label']) OVER();
```

Parameters

Argument	Description
mapName	The name of the mapping from which you are unloading the dictionary.
language	The language: <ul style="list-style-type: none">• 'english' or 'en'• 'spanish' or 'es'
label	The label of the mapping that you want to unload.

Examples

```
select unloadLabeledMapping(standard USING PARAMETERS label='custom_mapping') OVER();  
  
success  
-----  
t  
(1 row)
```

Best Practices for OEM Customers

Welcome to Best Practices for Vertica OEM Customers. This guide describes best practices and tips for Vertica OEM customers who embed their Vertica database with their own application and deliver it as an on-the-premises solution. Typically, the end users of the application do not interact with the Vertica database directly but rather through the application.

The information in this guide describes technical issues for two types of tasks that OEM customers must perform to optimize the database in their application:

Developing and testing the Vertica database:

- Running Database Designer and designing projections
- Managing your system resources
- Managing data load
- Optimizing delete and update operations
- Dropping partitions to save disk space

Packaging the Vertica database with your application:

- Installing Vertica silently
- Configuring the Vertica database using parameters defined during the development phase
- Using the Administration Tools command-line interface to create, start, and connect to the Vertica database

About This Document

This document outlines best practices and tips for Micro Focus OEM customers.

OEM customers embed the Vertica database with their own application and deliver it as an on-the-premises solution. Typically, the end users of the application do not interact with the Vertica database directly; they interact with the database only through the application.

The information in this document addresses technical issues for three types of tasks that Micro Focus OEM customers must perform to optimize the database in their application:

Part 1: Developing and Testing the Vertica Database

During your application development, install a Vertica database and load it with sample data. You can simulate and test the behavior of the database and application in a production-like environment.

The following sections describe the steps to take and the Vertica features to focus on for the Vertica database to work as desired with your application at a customer site:

- [Installing the Vertica Database](#)
- [Identifying Typical Queries](#)
- [Loading Representative Data](#)
- [Optimizing the Database with Database Designer](#)
- [Managing Database Resources](#)
- [Loading Data into the Database](#)
- [Dropping Partitions Due to Space Constraints](#)
- [Delete and Update Operations](#)

Installing the Vertica Database

To install the Vertica software on your development system, follow the instructions in [Installing Vertica](#).

Loading Representative Data

You can effectively test the way the Vertica database interacts with your application in a production environment. To do so, load data that is representative of the data that you might use in a production environment. Representative data allows you to effectively test database functionality and performance.

For more information about loading data into a Vertica database, see [Bulk-Loading Data](#) in the Administrator's Guide.

Identifying Typical Queries

Identify common queries that your application uses. You can provide these queries to Database Designer, Database Designer to create projections that maximize performance throughput in your database.

Optimizing the Database with Database Designer

An Vertica database stores its data in a physical format called *projections*. For optimal performance, projections should be optimized based on data and workload characteristics.

As Vertica OEM customers, optimize the projection design during your development cycle. You can standardize on that design for all deployments of your application.

The following sections describe the process of tuning projections using Database Designer:

- [What Does Database Designer Do?](#) (Administrator's Guide) gives an overall view of what Database Designer does for your database.
- [Results of Running Database Designer](#) (Administrator's Guide) explains what benefits your database achieves by running Database Designer.
- [Exporting the Optimized Database](#) (in this guide) contains information about how to deploy the optimized projections when the database is embedded with your application.

After you create your database for the first time, load the typical queries and sample data on your development system. Then, create a comprehensive database design as described in [Comprehensive Design](#) in the Administrator's Guide.

What Does Database Designer Do?

Database Designer uses sophisticated strategies to create physical schemas or database designs. These designs provide excellent query performance while using disk space efficiently.

Database Designer can create two types of designs:

- A comprehensive design allows you to create new projections for all tables in your database.
- A query-specific design creates projections for all tables referenced in the queries you supply to Database Designer.

Database Designer accepts unlimited sample queries for a comprehensive design, It can accept up to 100 sample queries in the query input file for a query-specific design.

Results of Running Database Designer

In most cases, the projections that Database Designer creates provide excellent query performance within physical constraints while using disk space efficiently. Database Designer:

- Recommends buddy projections, which significantly improve load, recovery, and site node performance.
- Automatically rebalances data after you add or remove nodes.
- Produces higher quality designs by considering [UPDATE](#) and [DELETE](#) statements.

Database Designer creates two scripts:

- Design script—Contains all the DDL statements that are needed to create the design.
- Deployment script—Contains all the CREATE PROJECTION scripts that are needed to create the projections for the design. The deployment script is a subset of the design script.

[Exporting the Optimized Database](#) in this guide describes how to configure these scripts so that you can standardize all customers on the optimal projections. The scripts must be portable to deployments with different numbers of nodes and K-safety levels.

Exporting the Optimized Database

After you optimize the database using Database Designer and other techniques, verify that the optimized projections are available for all customers in the deployed database.

The following Vertica functions generate scripts that create the database on the customer's cluster:

- [EXPORT_CATALOG](#)—Generates a SQL script that re-creates a physical schema design in its current state on a different cluster.
- [EXPORT_OBJECTS](#)—Generates a SQL script that re-creates on a different cluster all the non-virtual objects to which you have access. Running the generated SQL script on another cluster creates all referenced objects before their dependent objects.
- [EXPORT_TABLES](#)—Generates a SQL script that re-creates a logical schema on a different cluster. This logical schema includes schemas, tables, constraints, and view.

After you have created these scripts, edit them as follows:

- Modify the CREATE PROJECTION statements to include the appropriate segmentation clauses.
- Modify the CREATE PROJECTION statements to specify ALL NODES KSAFE *<value>*. As a result, all unsegmented projections are replicated on all nodes, with the designated K-safety value. If you don't specify a K-safety value, Vertica uses the current K-safety value, or 1.

Managing Database Resources

For large data sets of complex workloads, Micro Focus recommends that you run the database software on dedicated servers.

In some installations, you may have to run the Vertica database on a shared server with your application. To manage the CPU and memory usage so that neither the database nor your application monopolizes them, consider the following tips:

- [Configuring CPU Resources](#)
- [Configuring Memory Resources](#)

Configuring CPU Resources

You can configure CPU resources in two ways:

- Pinning Vertica processes to certain CPUs
- Setting run-time priorities for resource pools

Pinning Vertica Processes to Certain CPUs

To pin an entire Vertica server process and its child processes to execute on a specified set of CPUs in the cluster, use the following configuration parameters. These parameters do not affect external processes and applications:

- `PINPROCESSORS`: Number of CPUs to which Vertica processes will be pinned
- `PINPROCESSORSOFFSET`: Offset for the start of CPUs to be used by the Vertica server

For example, suppose you have a cluster where each node has 24 CPUs, numbered 0 through 23. In this case, you can restrict the server process to CPUs #16–23 by entering the following statements:

- `SELECT SET_CONFIG_PARAMETER('PINPROCESSORS', 8)`
- `SELECT SET_CONFIG_PARAMETER('PINPROCESSORSOFFSET', 16)`

Pin your application to the processors that are not allocated to the Vertica database. To make sure that other applications do not use the processors assigned to the Vertica server, use the Linux `cgroups` command.

Setting Run-Time Priorities for Resource Pools

You can manage resources in a resource pool. For you to do so, those resources must be assigned to queries already running in that resource pool.

You assign each resource pool a run-time priority of `HIGH`, `MEDIUM`, or `LOW`. These settings determine the amount of run-time resources (such as CPU and I/O bandwidth) assigned to queries in the resource pool when they run. Queries in a resource pool with a `HIGH` priority are assigned greater run-time resources than those in resource pools with `MEDIUM` or `LOW` run-time priorities.

Configuring Memory Resources

To limit the amount of overall system memory that you allocate to the Vertica database, set the `MAXMEMORYSIZE` parameter on the `GENERAL` resource pool.

To prevent swapping, verify that your application uses only the memory that the database is not using.

For example, suppose the `MAXMEMORYSIZE` of the `GENERAL` pool for the database is set to 32

GB on a system with 48 GB of RAM. In this case, your application should use no more than $(48 - 32) = 16$ GB of RAM.

Loading Data into the Database

While your application is running, you load data into your Vertica database on a regular basis. You can collect large amounts of data to load into your database at once, if you can meet your latency service-level agreement (SLA).

Simulate your data-loading operations in a production environment. Make sure to replicate the maximum expected rates of loading and spiking.

The following sections describe how Vertica stores data and how to load data into your database. They also describe how you can tune the processes and monitor the system to prevent performance problems:

- [WOS, ROS, and the Tuple Mover](#)
- [How Loading Works](#)
- [Loading Data from a File](#)
- [Loading Data from Your Application](#)
- [Parallel Load Streams](#)
- [Trickle Loading](#)
- [Monitoring Load Operations](#)

WOS, ROS, and the Tuple Mover

Vertica supports INSERT, UPDATE, DELETE, and bulk load operations (COPY), intermixed with queries in a typical data warehouse workload. The storage model consists of three elements, which operate the same on each Vertica node:

- *Write Optimized Store (WOS)* is a memory-resident data structure for storing INSERT, UPDATE, DELETE, and COPY (without `/*+DIRECT*/` hints) actions. To support very fast data load speeds, the WOS stores records without data compression or indexing. The WOS organizes data by epoch and holds both committed and uncommitted transaction data.

- *Read Optimized Store (ROS)* is a highly optimized, read-oriented, disk storage structure. The ROS makes heavy use of compression and indexing. You can use the `COPY...DIRECT` and `INSERT` (with `/*+DIRECT*/` hints) statements to load data directly into the ROS.
- *The Tuple Mover (TM)* is the Vertica database optimizer component that moves data from memory (WOS) to disk (ROS). The Tuple Mover runs in the background, performing some tasks automatically at time intervals determined by its configuration parameters.

For further information about the WOS, ROS, and Tuple Mover, see [Hybrid Storage Model](#) in Vertica Concepts.

How Loading Works

When you load data into a database, Vertica first moves it into memory (WOS) by default. You can override this default, and specify that Vertica move the data directly to disk (ROS). Periodically, the Tuple Mover:

- Moves data from the WOS to the ROS.
- Combines like-size ROS containers.
- Deletes purges records.

If the you move too much data too frequently to the WOS, the data can spill to disk. No data is lost, but such spillover can cause performance problems.

Loading Data from a File

If the data to be loaded into your database is stored in a file, use the `COPY` statement to load the data. You can also use `COPY` to load data from a data stream.

In its basic form, use `COPY` as follows:

```
COPY target-table FROM data-source
```

For more information, see the following sections of the Vertica documentation:

- [Using COPY and COPY LOCAL](#) in the Administrator's Guide
- [COPY](#) in the SQL Reference Manual
- [COPY LOCAL](#) in the SQL Reference Manual

Loading Data from Your Application

If the application needs to push data into the Vertica database, use the driver associated with the programming language that your application uses. For more information, see:

- [Loading Data through ODBC](#) in Connecting to Vertica
- [Loading Data through JDBC](#) in Connecting to Vertica
- [Loading Data through ADO.NET](#) in Connecting to Vertica

Parallel Load Streams

When you have a large amount of data to load, use parallel load streams to distribute the load operations across the cluster. To do this, create threads from multiple nodes that are connected to the Vertica database and load the data. This approach lets you use vsql, ODBC, ADO.NET, or JDBC. You can load server-side files or client-side files using the COPY from LOCAL statement.

Best practices for parallel load streams are:

- Issue a single multi-node COPY command that loads different files from different nodes. Specify the *nodename* option for each file.
- Issue a single multi-node COPY command that loads different files from any node using the ON ANY NODE option.

For additional information, see [Using Parallel Load Streams](#) in the Administrator's Guide.

Trickle Loading

Trickle loading is the process of loading data into your database periodically. You can use trickle loading to create a more sustainable load process that generates fewer ROS containers.

The Vertica database uses the transaction isolation level of READ COMMITTED. This level allows users to see the most recently committed data without holding any locks.

READ COMMITTED also allows new data to be loaded while concurrent queries are running.

To tune the trickle-loading operations:

- Adjust batch size. Generally, larger load batches are more efficient.
- Alter the moveout/mergeout parameters:
 - MergeOutInterval
 - MoveOutInterval
 - MoveOutMaxAgeTime
 - MoveOutSizePct
- Limit the load process to a specific resource so that other resources are available to queries. To do so, create a resource pool specifically for the trickle-loading process.

Monitoring Load Operations

When you simulate the load operations while developing your application, there are two system tables you must monitor. These tables help you verify that you have configured load operations in a way that negative effects on the database or application performance:

- [PROJECTION_STORAGE](#)
- [PARTITIONS](#)

PROJECTION_STORAGE

While developing your application, monitor the ROS count over time. To monitor the ROS count:

- Query the ROS_COUNT field in the PROJECTION_STORAGE system table.
- Query partitions that show the ROS_COUNT per partition.

The default maximum for the ROS_COUNT field is 1024. The ROS_COUNT value should not come close to 1024 for the duration of a partition and through the time it takes an old partition to merge. If the ROS_COUNT field does not come close to 1024, your loading techniques should be adequate.

```
VMart=> SELECT projection_id, projection_name, ros_count
         FROM projection_storage;
 projection_id | projection_name | ros_count
-----+-----+-----
45035996273719430 | promotion_dimension_super | 1
```

```
45035996273720042 | online_page_dimension_super | 1
45035996273719788 | employee_dimension_super   | 1
45035996273719276 | store_dimension_super       | 1
45035996273719610 | customer_dimension_super    | 1
45035996273722132 | T_super                      | 1
45035996273719098 | product_dimension_super     | 1
45035996273719942 | warehouse_dimension_super   | 1
45035996273720000 | shipping_dimension_super    | 1
45035996273720498 | online_sales_fact_super     | 1
45035996273719536 | vendor_dimension_super      | 1
45035996273720206 | store_sales_fact_super      | 1
45035996273720652 | inventory_fact_super        | 1
45035996273718920 | date_dimension_super        | 1
45035996273720336 | store_orders_fact_super     | 1
45035996273720100 | call_center_dimension_super  | 1
45035996273722194 | T2_super                    | 1
(17 rows)
```

PARTITIONS

While developing your application, monitor the PARTITIONS system table. Make sure that the number of ROS containers per partition does not reach the maximum of 1024.

In addition, monitor the `ros_size_bytes` field to see if there are any small ROS containers that you can merge.

```
=> SELECT projection_name, ros_id, ros_size_bytes
   FROM partitions;
projection_name |      ros_id      | ros_size_bytes
-----+-----+-----
t_p            | 45035996273740461 |          90
t_p            | 45035996273740477 |          99
t_p            | 45035996273740493 |          99
```

Dropping Partitions Due to Space Constraints

If your hardware has fixed disk space, you might need to configure a regular process to roll out old data by dropping partitions.

For example, if you have only enough space to store data for a fixed number of days, configure Vertica to drop the partition with the oldest date. To do so, create a time-based job scheduler, such as `cron`, to drop the partition on a regular basis during low-load periods.

If the ingest rate for data has peaks and valleys, you can use two techniques to manage how you drop partitions:

- Set up a process to check the disk space on a regular (daily) basis. If the percentage of used disk space exceeds a certain threshold—for example, 80%—drop the oldest partition.
- Add an artificial column in a partition that increments based on a metric like row count. For example, that column might increment each time that the row count increases by 100 rows. Set up a process that queries that column on a regular (daily) basis. If the value in the new column exceeds a certain threshold—for example, 100—drop the oldest partition, and set the column value back to 0.

For more information about partitions, see the following sections in the Vertica documentation:

- [CREATE TABLE](#) in the SQL Reference Manual
- [Working with Table Partitions](#) in the Administrator's Guide

Load Balancing

In Vertica, load balancing supports multiple client connections through an IP address that is shared among all nodes in a cluster. Your cluster needs to balance incoming client requests across nodes and prevent node exclusion from clients in the case of node failure.

While you are developing your application, simulate a high load of activity in your cluster to mimic the expected behavior after the application is running.

You can perform load balancing on your database cluster using any of these methods:

- [Native Connection Load Balancing](#)
- [Third-Party Load Balancers](#)

For more detailed information, see the [Connection Load Balancing](#) and [Adding Nodes](#) sections in the Administrator's Guide.

Native Connection Load Balancing

The Vertica server and client libraries have native connection load balancing built in. This feature distributes client connection CPU and memory overhead across all nodes in the database cluster. Native connection load balancing can prevent overburdening some nodes with many client connections, while other nodes have few client connections.

You must enable native connection load balancing on both the Vertica server and the client.

For more information about native connection load balancing, see the following topics in the Administrator's Guide:

- [About Native Connection Load Balancing](#)
- [Enabling and Disabling Native Connection Load Balancing](#)
- [Monitoring Native Connection Load Balancing](#)

To learn how to enable native connection load balancing with your specific client software, see the following topics in Connecting to Vertica:

- [Enabling Native Connection Load Balancing in JDBC](#)
- [Enabling Native Connection Load Balancing in ODBC](#)
- [Enabling Native Connection Load Balancing in ADO.NET](#)

Third-Party Load Balancers

You can perform load balancing across your Vertica database cluster using third-party load balancers.

Third-party load balancers exist as software applications or hardware appliances. They manage the inbound and outboard data traffic and network connections throughout the cluster. Their goal is to distribute the workload evenly among all the nodes in the cluster.

Many third-party load balancers are available.

Before choosing a load balancer for your Vertica database cluster, consider these configuration characteristics:

- Size of the cluster
- Potential future growth of your cluster
- Security
- Performance

Delete and Update Operations

You may have small configuration tables with more of an online transaction processing (OLTP) workload. If so, your Vertica database can handle this situation efficiently.

The following sections explain how to configure your database to ensure optimal performance during delete and update operations:

- [How Vertica Deletes and Updates Data](#)
- [Buffering Deletes and Updates](#)
- [Monitoring Deletes and Updates](#)
- [Purging Deleted Data](#)

How Vertica Deletes and Updates Data

In Vertica, delete operations do not actually remove rows from physical storage. Instead, the `DELETE` statement marks rows as deleted.

Update operations have the same effect; the `UPDATE` statement actually combines delete and insert operations. The deleted rows remain in physical storage until the Tuple Mover performs a purge operation. The purge operation permanently removes deleted data so that the disk space can be reused.

For more information about how deletes and updates work in Vertica, see [Mergeout](#) in the Administrator's Guide.

Buffering Deletes and Updates

The most efficient way you can perform delete and update operations with Vertica is to update many database records at a time using fewer statements. To do this, buffer database deletes and updates in your application before you submit them. That way, you can manage the flow of the delete and update operations so that they do not interfere with normal database performance.

For more information, see [Best Practices for DELETES and UPDATES](#) in the Administrator's Guide.

Monitoring Deletes and Updates

The frequency of delete and update submissions to the database should be sustainable and not negatively impact performance. While you develop your application, make sure to monitor the delete and update operations in the database. To do so, monitor the `DELETE_VECTORS` system table. This table provides a view of the number of deleted records and their corresponding delete vectors that are still physically stored in the system:

```
VMart=> SELECT node_name, schema_name, projection_name,
              deleted_row_count, used_bytes FROM delete_vectors;
-[ RECORD 1 ]-----+-----
node_name      | v_vmart_node0001
schema_name    | public
projection_name | product_dimension_super
deleted_row_count | 32345
used_bytes     | 524288
-[ RECORD 2 ]-----+-----
node_name      | v_vmart_node0001
schema_name    | public
projection_name | employee_dimension_super
deleted_row_count | 943
used_bytes     | 54788
-[ RECORD 3 ]-----+-----
node_name      | v_vmart_node0001
schema_name    | public
projection_name | warehouse_dimension_super
deleted_row_count | 20
used_bytes     | 16384
```

Purging Deleted Data

Consider scheduling a nightly purge of small tables that receive frequent updates. This operation rewrites the tables while removing deleted records. Run the nightly purge against small tables that would not be expensive to rebuild and that have many deleted records.

For more information, see [PURGE_TABLE](#) in the SQL Reference Manual.

Part 2: Packaging the Vertica Database with Your Application

After you install your Vertica database, load sample data into it, and test the database operation in a simulated production environment. Then, create a script that installs and configures the database with your application. This script should

- Perform its operations silently, in a way that is invisible to your end users.
- Be portable so that it can install the database on a cluster with any number of nodes.

The following sections describe what functions the script should perform.

- [Installing the Vertica Database at the Command Line](#)
- [Creating the Database](#)
- [Configuring the Database](#)
- [Starting the Database](#)
- [Connecting to the Database](#)
- [Backing Up the Database](#)

Installing the Vertica Database at the Command Line

The packaging script should contain commands to execute the Vertica installation procedure.

Important: Vertica must be installed separately for each user. Do not create a system image for subsequent installations, which may cause all users to use the same SSH keys. This situation can result in serious security issues.

The commands in the packaging script should perform the following tasks:

1. Login as the root user.
2. Create a properties file that enables non-interactive setup by supplying the parameters you want the Vertica database to use. The properties file is described in [Installing Vertica Silently](#) in Installing Vertica.
3. Install the database by running the `install_vertica` script, as described in [Installing Vertica Silently](#) in Installing Vertica.
 - a. Make sure to run the scripts with the `-y` parameter to avoid the script asking for EULA agreement.
 - b. On a single-node installation, consider using the host name or IP address instead of `localhost` or `127.0.0.1` because of the following limitations:
 - The `-s` parameter is required for multi-node installations. On single-node installations the parameter is optional, and the default is `localhost`. However, if you plan to expand to additional hosts later, you must use the `-s` parameter. If you do not use the `-s` parameter, your Vertica installation cannot be upgraded to a multi-node deployment.
 - On a single-node `localhost` installation, the installer does not set up a passwordless `ssh`. Vertica's backup scripts require that the administrator be able to log into the node via `ssh` without a password. Consequently, if you want to use the backup scripts, you must manually enable passwordless `ssh` logins for any single-node installation.
 - If either side of a connection is a single-node cluster installed to `localhost`, importing and exporting data fails. Failure also occurs if you do not specify a host name or IP address for your single-node cluster.
4. To save any warnings that the `install_vertica` script returns, redirect any warning messages to a separate file. To do so, specify the `redirect_output = filename` parameter in the properties file. After the installation completes, review the warnings and correct any problems that are critical.
5. If you need to access the Vertica database from a client application, follow the steps in [Installing the Client Drivers](#) in Connecting to Vertica.
6. If you need to use the Vertica `vsq` executable image on a non-cluster Linux host to connect to an Vertica database, follow the steps in [Optionally Install the vsq Client Application on Non-Cluster Hosts](#) in Installing Vertica.

7. Disconnect from the Administration Host.
8. Unless you used the `-L` parameter in the properties file, install the license key you downloaded.

Note: To use data or catalog directories other than those that the `install_vertica` script creates, create those directories, and make sure that `dbadmin` owns them.

Creating the Database

Before you embed the Vertica database with your customer application, you must create it. To do so, add the following command to the packaging script:

```
$ /opt/vertica/bin/admintools --tool create_db [ options ]
```

Command-Line Options:

Option	Function
<code>-h</code> <code>--help</code>	Show this help message and exit.
<code>-s NODES</code> <code>--hosts=NODES</code>	Comma-separated list of hosts to participate in database.
<code>-d DB</code> <code>--database=DB</code>	Name of database to be created.
<code>-c CATALOG</code> <code>--catalog_path=CATALOG</code>	[Optional] Path of catalog directory.
<code>-D DATA</code> <code>--data_path=DATA</code>	[Optional] Path of data directory.
<code>-p DBPASSWORD</code> <code>--password=DBPASSWORD</code>	[Optional] Database password in single quotes.
<code>-l LICENSEFILE</code> <code>--license=LICENSEFILE</code>	[Optional] Database license.
<code>-P POLICY</code> <code>--policy=POLICY</code>	[Optional] Database restart policy.

The following example shows how to create a database named `mydb`:

```
$ admintools -t create_db
-s 10.20.100.66,10.20.100.67,10.20.100.68
-d mydb -c /home/dbadmin/mydb/catalog
-D /home/dbadmin/mydb/data -l /home/dbadmin/vlicense.dat

Info: no password specified, using none
Distributing changes to cluster.
10.20.100.66 OK [vertica][(6, 1, 2)][20130430][x86_64]
10.20.100.67 OK [vertica][(6, 1, 2)][20130430][x86_64]
10.20.100.68 OK [vertica][(6, 1, 2)][20130430][x86_64]
Checking full connectivity
Creating database mydb
Node Status: v_mydb_node0001: (DOWN)
Node Status: v_mydb_node0001: (INITIALIZING)
Node Status: v_mydb_node0001: (VALIDATING LICENSE)
Node Status: v_mydb_node0001: (UP)
Creating database nodes
Creating node v_mydb_node0002 (host 10.20.100.67)
Creating node v_mydb_node0003 (host 10.20.100.68)
Node Status: v_mydb_node0001: (UP) v_mydb_node0002: (UP)
v_mydb_node0003: (DOWN)
Node Status: v_mydb_node0001: (UP) v_mydb_node0002: (UP)
v_mydb_node0003: (UP)
Database mydb created successfully.
```

Configuring the Database

Execute the scripts that you created in [Exporting the Optimized Database](#) in this guide that:

- Create the physical schema.
- Create the catalog objects.
- Create the logical schema.
- Create projections that have been configured for use with the application.

Starting the Database

To start your Vertica database, add a command to your packaging script that uses the `admintools` script with either the `start_db` or `restart_db` option:

- The `start_db` option starts a database/
- The `restart_db` option allows you to restart the database at the last good epoch, minimizing any loss of data.

start_db Option

```
$ /opt/vertica/bin/admintools --tool start_db [ options ]
```

Command-Line Options

Option	Purpose
-h --help	Show this help message and exit.
-d <i>DB</i> --database= <i>DB</i>	Name of database to be started.
-p <i>DBPASSWORD</i> --password= <i>DBPASSWORD</i>	Database password in single quotes.
-i --noprompts	Do not stop and wait for user input. (Default: False.)

The following example starts the database called VMart and assigns the database password Vertica1:

```
$ admintools -t start_db -d VMart -p Vertica1
```

restart_db Option

```
$ /opt/vertica/bin/admintools --tool restart_db [ options ]
```

Command-Line Options

Option	Purpose
-h --help	Show this help message and exit.
-d <i>DB</i> --database= <i>DB</i>	Name of database to be restarted.
-e <i>EPOCH</i> --epoch= <i>EPOCH</i>	Epoch at which the database is to be restarted. If 'last' is given as the argument, the database restarts from the last good epoch.
-p <i>DBPASSWORD</i> --password= <i>DBPASSWORD</i>	Database password in single quotes.

Option	Purpose
-i --noprompts	Do not stop and wait for user input. (Default: False.)

Connecting to the Database

To connect to the Vertica database that you started, add the following command to the packaging script:

```
$ /opt/vertica/bin/admintools --tool connect_db [ options ]
```

Command-Line Options

Option	Purpose
-h --help	Show this help message and exit.
-d <i>DB</i> --database= <i>DB</i>	Name of database to connect to.
-p <i>DBPASSWORD</i> --password= <i>DBPASSWORD</i>	Database password in single quotes.

Backing Up the Database

Important: Micro Focus strongly recommends backing up the database on a regular basis. This task is especially important for a single-node environment where K-safety = 0.

Micro Focus provides a comprehensive utility, the `vbr` Python script, that lets you back up, restore, list backups, and copy your database. You can create full and incremental database backups of specific schemas or tables to use with a multi-tenant database. Using `vbr`, you can save your data to a variety of locations:

- A local directory on the nodes in the cluster
- One or more hosts outside of the cluster
- A different Vertica cluster (in effect, cloning your database)

Create regular backups of your data by running `vbr` from a cron job or other task scheduler.

Part 3: Ongoing Database Maintenance

After your application is running at the customer site, monitor your database performance in the following areas. These steps help you prevent performance issues and down time:

- [Monitoring System Resources for Concurrent Use](#)
- [Monitoring Database Size](#)
- [Updating Statistics](#)
- [Optimizing the Physical Schema](#)
- [Optimizing Query Performance](#)
- [Database Recovery](#)
- [Hardware Maintenance](#)
- [Troubleshooting Performance](#)

Monitoring System Resources for Concurrent Use

When running in a multi-user environment, you may have several processes competing concurrently for system resources. Use the built-in GENERAL resource pool to provide resources in a multi-user environment. The GENERAL pool is preconfigured based on your system's RAM and machine cores. You can customize the GENERAL pool or define new resource pools and configure them for memory usage, concurrency, and query priority.

To monitor use of system resources over time, query the following system tables.

System Table Name	Description
RESOURCE_ACQUISITIONS	Details about each resource (memory, open file handles, threads) acquired by each request for each resource pool in the system.
RESOURCE_POOL_DEFAULTS	Default values for parameters in each internal and user-defined resource pool.

System Table Name	Description
RESOURCE_POOL_STATUS	Configuration settings of the various resource pools in the system, including internal pools.
RESOURCE_POOLS	Information about settings with which each resource pool was configured.
RESOURCE_QUEUES	Information about requests that are pending for various resource pools.
RESOURCE_REJECTIONS	Information about requests for resources that are rejected by the Resource Manager.
RESOURCE_REJECTION_DETAILS	Entries for each resource request that the Vertica database denies. This information is useful for determining if there are resource space issues and which users/pools encounter problems.
SYSTEM_RESOURCE_USAGE	History about system resources, such as memory, CPU, network, disk, I/O.

For more information about managing resource pools, see [Best Practices for Managing Workload Resources](#) in the Administrator's Guide.

Monitoring Database Size

You can use your Vertica database until the columnar data reaches the maximum raw data size that the license agreement specifies. In this context, *raw data* means the uncompressed data stored in a single database. The raw data size is estimated as if the data had been exported from the database in text format, rather than as compressed data.

The database uses statistical sampling to calculate an accurate estimate of the raw data size of your database. This approach allows Vertica to estimate the database size without significantly affecting database performance.

Some data is not included when calculating the database size:

- Multiple projections of data from a table. Data appearing in multiple projections of the same table is counted only once.
- Data stored in temporary tables.

- Data accessible through external table definitions.
- Data that has been deleted but not purge from the database.
- Data stored in the WOS.
- Data stored in system tables such as monitoring tables, Data Collector tables, and Database Designer tables.
- Delimiter characters.

If your database size approaches your licensed usage allowance, you see warnings in the Administration Tools and vsql. You have two options to eliminate these warnings:

- Upgrade your license to a larger data size allowance.
- Delete data from your database to remain under your licensed raw data size allowance.

If your database continues to grow, after a grace period, Vertica displays additional warnings in more parts of the system.

If your database size exceeds your licensed data allowance, all successful queries from ODBC and JDBC clients return with a status of `SUCCESS_WITH_INFO` instead of `SUCCESS`. The result contains a warning about your database size. Your ODBC and JDBC clients should be prepared to handle these messages instead of assuming that successful requests always return `SUCCESS`.

If you need a more accurate database size estimate than statistical sampling can provide, use the `AUDIT` function. The `AUDIT` function has parameters that allow you to tune its execution to minimize the impact on database performance.

For more information, see:

- [Calculating the Database Size](#)
- [Monitoring Database Size for License Compliance](#)
- [AUDIT](#)

Updating Statistics

The Vertica query optimizer relies on up-to-date statistics for tables, schemas, and the database. The statistics allow the optimizer to determine the most efficient plan to execute a query.

The following statistics can affect optimizer decisions:

- Eligible projections to answer the query
- The best order in which to perform joins
- Plans involving different algorithms, such as hash join/merge join or group by hash/group by pipelined operations
- Data distribution algorithms, such as broadcast and resegmentation

Without accurate statistics, the optimizer could choose a suboptimal projection or join order for a query.

As you update your data over time, the statistics become out of date. Stale statistics can cause the optimizer to choose a less-than-optimal plan for executing a query. The query plan that the EXPLAIN statement creates contains information about statistics.

How often you update statistics depends on how often your data changes. Micro Focus recommends that you schedule statistics analysis on a regular basis. Regular updates help the optimizer choose the most efficient plan for executing your queries.

For guidance on updating statistics, see [Collecting Database Statistics](#) in the Administrator's Guide.

Vertica provides the following functions related to collecting statistics.

Function	Description
ANALYZE_STATISTICS	Collects and aggregates statistics from all nodes that store projections associated with the specified table or column.
DROP_STATISTICS	Removes statistics for the specified table and lets you optionally specify the category of statistics to drop.
EXPORT_STATISTICS	Exports statistics to an XML file.
IMPORT_STATISTICS	Imports statistics from an XML file created by EXPORT_STATISTICS.

Optimizing the Physical Schema

After you load data into your database, first run Database Designer to create a physical schema that provides optimal query performance.

The first time you run Database Designer, make sure that Database Designer has representative queries and data on which to base the design. Then, create a comprehensive design.

Consider rerunning Database Designer to create an incremental design when:

- The data has changed significantly.
- You use different queries on a regular basis.
- You encounter performance issues with your queries.
- Your schema changes.

For more information, see [Creating a Database Design](#) in the Administrator's Guide.

Optimizing Query Performance

When you submit a query to Vertica for processing, the Vertica query optimizer automatically chooses a set of operations to compute the requested result. These operations together are called a *query plan*. The choice of operations can drastically affect the run-time performance and resource consumption needed to compute the query results. Depending on the properties of the projections defined in your database, the query optimizer can choose faster and more efficient operations to compute the query results.

As your database grows over time, queries could degrade in performance. It is important to monitor the performance of your queries—especially queries that are run frequently or access large data sets.

If you experience query performance issues, you should:

- Analyze the statistics and the histogram for your database tables and run Database Designer. Supply the queries that are experiencing performance problems.
- Review the QUERY_EVENTS system table. This table identifies whether there are issues with the planning phase of a query.

Database Recovery

Recovery is the process of restoring the database to a fully functional state after one or more nodes in the system has experienced a software- or hardware-related failure. Failures can

include:

- Hardware failure of one or more nodes
- Clean database shutdown (initiated)
- Unclean database shutdown, which can be caused by:
 - A critical node failure, leaving part of the data unavailable.
 - A power failure causing all nodes to reboot.
 - Vertica processes terminating due to a software or hardware failure.

A failure can cause a node to lose database objects or to miss DML changes (INSERTs, UPDATEs, and so on) while offline. During recovery, each node retrieves any lost objects and updates any changes by querying the other nodes.

For detailed information about database recovery after specific types of failure, see [Recovering the Database](#) in the Administrator's Guide.

Hardware Maintenance

For detailed information about the hardware required to run the Vertica database, see [Recommendation for Sizing Vertica Nodes and Clusters](#) on the [Vertica User Community](#).

For ongoing maintenance of your cluster nodes, see the next section [Adding or Removing Cluster Nodes](#).

Adding or Removing Cluster Nodes

You can scale your Vertica cluster by adding or removing nodes to meet the needs of your database. The most common way to scale up your cluster is to add nodes to accommodate more data and provide better query performance.

If your cluster has more nodes than it needs, or if you need to divert hardware for other uses, you can scale down your cluster.

You can add or remove nodes without having to shut down or restart the database.

After adding a node or before removing a node, the database rebalances the data. Rebalancing moves data around the cluster to populate the new nodes. It also moves data off the remaining nodes. Data can also be exchanged between nodes that are not being added or removed to

maintain robust intelligent K-safety. If too little disk space exists to rebalance in one iteration, the database rebalances iteratively.

For detailed information about managing your cluster, see [Managing Nodes](#) in the Administrator's Guide.

Troubleshooting Performance

If your application performance is slow, you need to determine whether your application or the Vertica software is causing the problem.

First, evaluate your system for performance issues. If you modify your application, but are still experiencing issues, evaluate your Vertica database for performance issues.

Vertica provides many tools to evaluate the performance of your database. The following sections of the Administrator's Guide describe them in detail:

- [Analyzing Workloads](#)
- [Using Diagnostic Tools](#)
- [Query Plans](#)
- [Profiling Database Performance](#)

The [Query Optimization](#) section of Analyzing Data gives information about improving the performance of your database queries.

Vertica Plug-In for Informatica

Welcome to the Vertica Plug-in for Informatica Guide.

Introduction to Using Informatica PowerCenter with Vertica

Informatica's PowerCenter family of products lets you collect, transform, and store data. The products support a wide variety of data sources, including databases, message queues, and many different file formats.

The PowerCenter Client consists of four main applications:

- Use Designer to create sources, targets, and mappings.
- Use Workflow Manager to create workflows for those sources, targets and mapping you created in Designer.
- Use Workflow Monitor to monitor running workflows.
- Use Repository Manager to manage repository resources, such as moving folders and objects and managing permissions and users.

The PowerCenter Server enables you to access, read, and write to Vertica.

PowerCenter 9.6.1 HotFix 2 includes the PowerExchange (PWX) Connector for Vertica.

Informatica developed this PWX Connector as an alternative to the Vertica plug-in.

The PWX connection includes additional capabilities and performance improvements when connected to Vertica.

History of Integration Between Vertica and Informatica

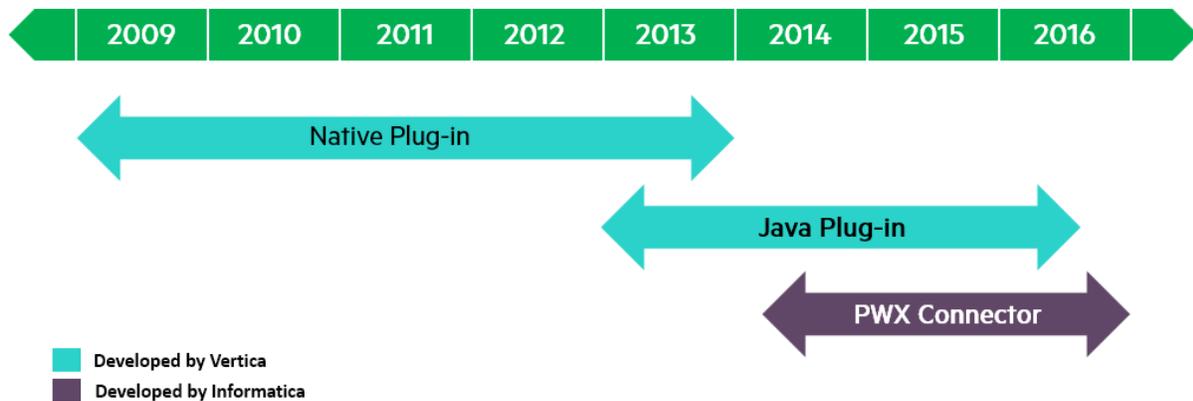
Prior to the PowerExchange (PWX) Connector for Vertica, Micro Focus developed and supported two connectors for Informatica and Vertica:

- In 2009, Micro Focus developed the Vertica Plug-in for Informatica. This connector used the native method of loading data from Informatica into Vertica.
- In 2013, Micro Focus replaced the earlier Vertica Plug-in for Informatica with a Java plug-in that supports all operating system platforms. This plug-in runs on generic JDBC and ODBC connections and includes new and improved features compared to the native plug-in.

This document describes how to use the Vertica Java plug-in for connecting from Informatica to Vertica.

In 2014, Informatica released the PWX Connector for Vertica. This connector includes enhancements to the partitioning and pushdown capabilities of Informatica.

The following timeline shows the history of plug-ins for connecting from Informatica to Vertica:

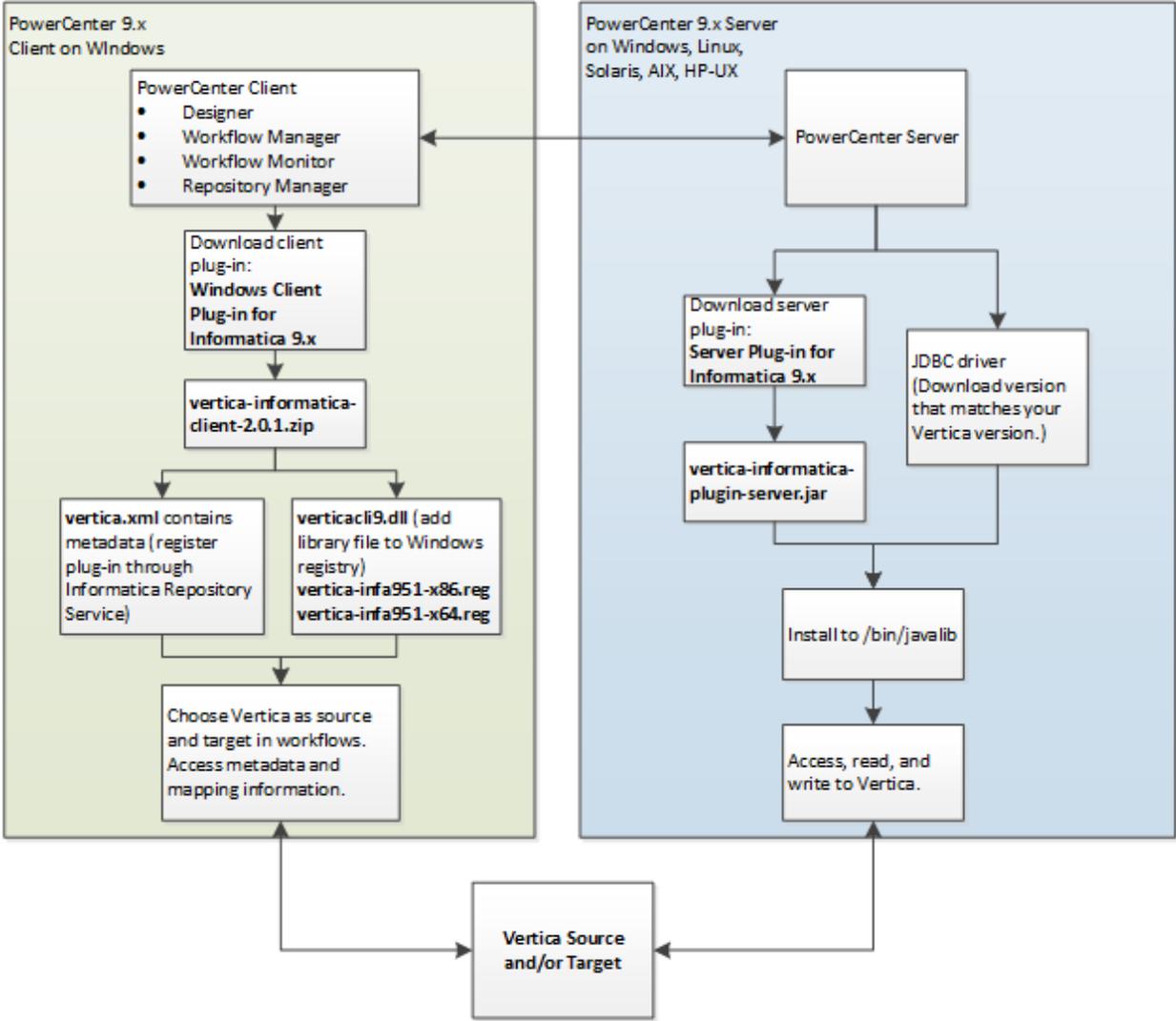


Micro Focus recommends that you use the new PWX Connector for Vertica with Informatica PowerCenter 9.6.1 HotFix 2 or later to connect to your Vertica database.

For detailed information about using the PWX Connector for Vertica, see [Vertica Integration with Informatica: Connection Guide](#).

How the Plug-in Is Configured with Vertica and Informatica PowerCenter

The following illustration provides an overview of the configuration.



This manual provides information for installing plug-in components, using the plug-in to access Vertica as source or target database, and implementing and modifying plug-in features.

Overview

The following table provides recommended steps.

Step	Action	Notes
1	Follow the procedures in Installing the Vertica Plug-in for PowerCenter to download and install plug-in	You must install both client and server components.

Step	Action	Notes
	components.	
2	Review the sample for using the plug-in with PowerCenter in Using the Vertica Plug-in with Informatica PowerCenter .	The sample shows how to import Vertica as source and target. It also provides steps that show you how to specify and connect to your Vertica database and include it in your workflows.
3	Set plug-in features according to your specific needs. Features are listed and described in Accessing and Setting Plug-in Features .	Be aware of situations where the use of one feature depends on another. For example, to take advantage of the increased performance of EnableStreamingBatchInsert , you must set Copy Local Method to None .
4	Check Best Practices for required memory settings and other tips.	Memory requirements are highly dependent upon Informatica and Vertica dedicated resources.

Installing the Vertica Plug-in for PowerCenter

You must download and install a client and a server component for the Vertica Plug-in for Informatica.

As a first step, download both the client and server components of the plug-in from the myVertica portal.

The client portion of the plug-in (`vertica-informatica-client-2.0.1.zip`) includes the following files:

- `vertica.xml` contains the metadata definition needed by the PowerCenter repository to allow communication between PowerCenter and Vertica.
- `verticacli9.dll` is a library file you add to your Windows registry.
- `vertica-infa951-x86.reg` is a registry file you can use to register the dll on 32-bit machines for Informatica PowerCenter 9.5.1.
- `vertica-infa951-x64.reg` is a registry file you can use to register the dll on 64-bit machines for Informatica PowerCenter 9.5.1.

The server portion includes the file `vertica-informatica-plugin-server.jar`.

Note: Each server type requires the Java 6.0 run-time environment.

Installing the Vertica plug-in is a multi-step process. The following sections explain these steps in greater detail, using a simple example.

1. Register the plug-in's metadata with the PowerCenter Repository Service with which you want to access Vertica. Follow the procedures in [Registering the Plug-in's Metadata](#).
2. Add the `verticacli9.dll` library file to the Windows registry. Follow the procedures in [Adding the Library File to the Windows Registry](#).
3. Copy the server plug-in to the PowerCenter server `java1ib` directory. Follow the procedures in [Copying the Plug-in Library on the Server](#).

Registering the Plug-in's MetaData

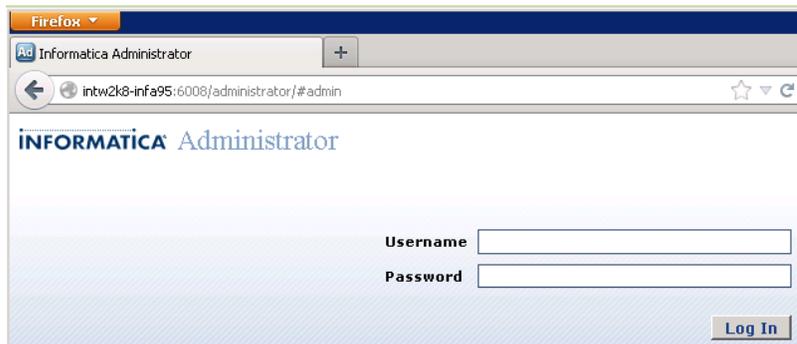
The PowerCenter repository needs information about the Vertica plug-in in order to enable clients to use it. This information is supplied in an XML-format file named `vertica.xml`.

Perform the following to register the plug-in's metadata.

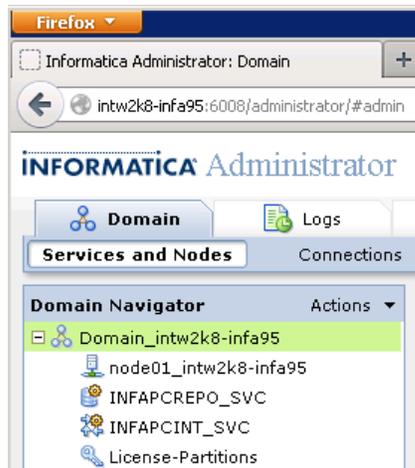
Switch to Exclusive Mode

Before you can register the plug-in's metadata, you must logon to PowerCenter and switch to exclusive mode to ensure that the repository does not change while you are registering the plug-in.

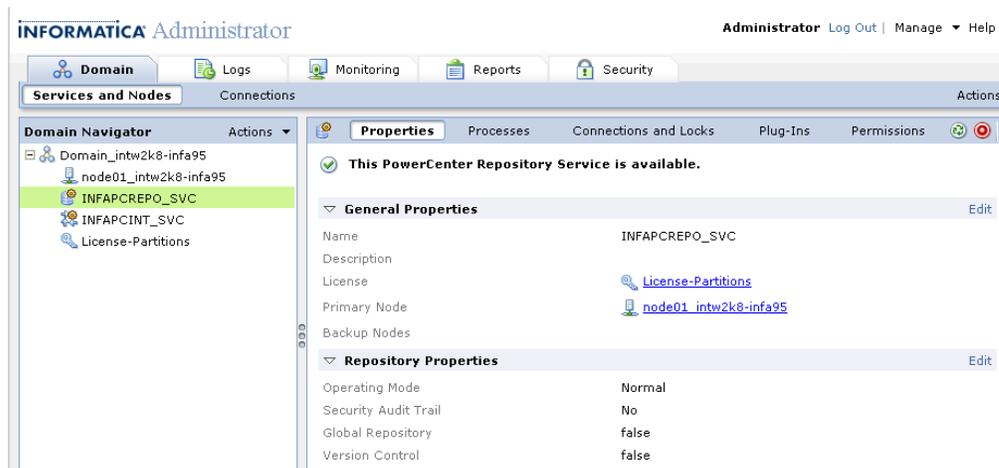
1. Place or copy the file `vertica.xml` to your system.
2. Open a browser and log into the PowerCenter domain's Administration Console.



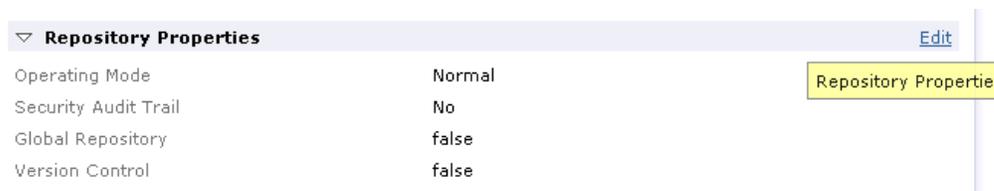
3. Select the **Domain** tab, and click **Services and Nodes**.



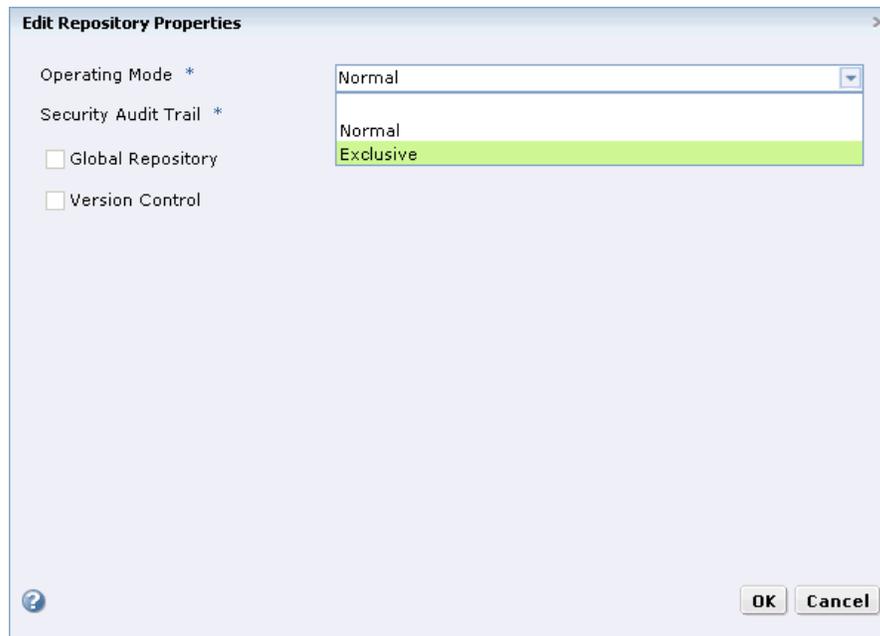
4. In the **Domain Navigator**, click the entry for the repository that you want to connect to Vertica.



5. Under the **Properties** tab, click **Edit** to edit the **Repository Properties** section.



6. In the **Operating Mode** list box, choose **Exclusive**, and then click **OK**.



7. In the **Restart Service** prompt, click **OK** to confirm switching to exclusive mode.



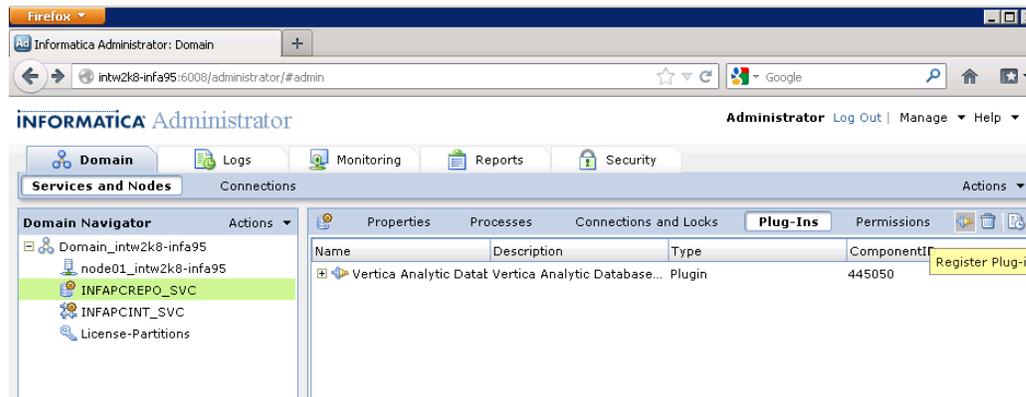
The Repository Service may take from a few moments to several minutes to restart and re-enable itself.

8. Wait until you see the notice, **This PowerCenter Repository Service is available.**



Register the Plug-in

1. On the **Plug-ins** tab, click the icon for registering a plug-in.



2. On the **Choose a plug-in** field, click **Browse** and select the `vertica.xml` in the folder where you earlier placed the file.



3. Under the **Repository Authentication** section, enter your repository username and password.

Note: If you are upgrading from a previously installed plug-in version, select the checkbox, **Update existing plug-in registration**. Otherwise leave the box cleared.

4. Click **OK** to upload the metadata file. The PowerCenter Administration Console uploads the metadata file and registers the Vertica plug-in data.

Switch Back to Normal Mode

1. On the Properties tab's **Repository Properties** section, click **Edit**.
2. In the **Operating Mode** list box, choose **Normal**.
3. In the Restart Service prompt, click **OK** to confirm switching to normal mode.

The Repository Service may take from a few moments to several minutes to restart and re-enable itself.

Adding the Library File to the Windows Registry

For each PowerCenter client system that you want to use with Vertica, you must install a copy of the `verticacli9.dll` file in the client binary folder. This folder is named `client\bin` in the PowerCenter install directory. The following path is typical of a PowerCenter installation for Informatica version 9.5.1:

```
C:\Informatica\9.5.1\clients\PowerCenterClient\client\bin
```

Copy and Register the `verticacli9.dll`

Copy the library file to the client binary directory (i.e., `client\bin`).

Then, add a registry entry to the Windows registry. Adding this entry tells the PowerCenter Designer to load the plug-in library. Perform one of the following to register the plug-in library.

Note: The registry file is specific to Informatica PowerCenter version 9.5.1. The Vertica Plug-in for Informatica has only been tested with this version. If you want to try to use it with another version of PowerCenter, you will need to manually add configuration information to the Windows registry, as explained below.

Register the `verticacli9.dll` Using a Registry File for Informatica PowerCenter 9.5.x

1. Double-click the registry file in Windows Explorer:
 - Use `vertica-infa951-x86.reg` to register the dll on 32 bit machines.
 - Use `vertica-infa951-x64.reg` to register the dll on 64 bit machines.
2. When asked if you want to add the contents of the file to the registry, click **Yes**.

Register the `verticacli9.dll` Manually for all Other Versions of Informatica (9.6.x, 10.x, et. al.)

1. Start the registry editor by typing `regedit.exe` in the Windows Start menu's command run command box.
2. Navigate to the correct location in the registry.

For 32-bit versions of Windows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Informatica\  
PowerMart Client Tools\x.x.x\Plugins\Informatica
```

For 64-bit versions of Windows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\  
Informatica\PowerMart Client Tools\x.x.x\  
Plugins\Informatica
```

Where `x.x.x` is the version of Informatica you are using (for example 9.5.1).

3. Right-click in the right pane of the Registry Editor window. Select **New** then select **String Value**.
4. Change the name of the string value from New Value #1 to `VERTICA`.
5. Double-click the new `VERTICA` entry. When prompted for a new value, enter `verticacli9.dll`.
6. Exit the registry editor.

Copying the Plug-in Library on the Server

The final step in setting up the Vertica plug-in for Informatica is to copy the Vertica server-side plug-in file to the proper directory on the PowerCenter server.

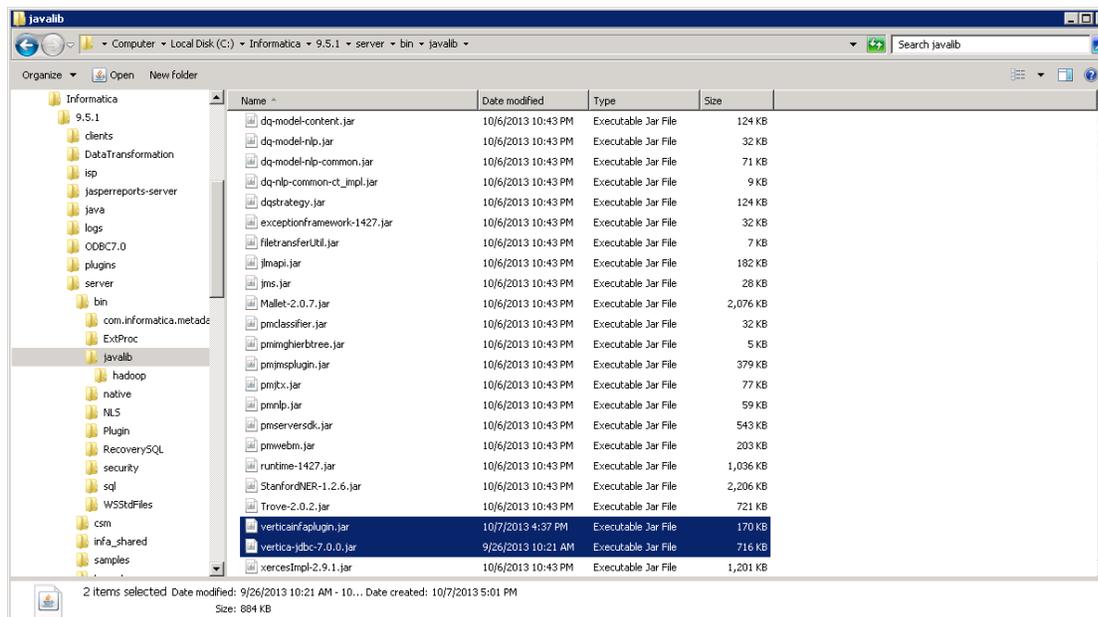
- `vertica-informatica-plugin-server.jar` (for both 32 and 64 bit servers)

Copy the library file to your server's binary directory, which is the `\bin\javallib` subdirectory in the PowerCenter server install directory. The full path to this directory for Windows is usually:

```
C:\Informatica\9.5.1\server\bin\javallib
```

Note: In addition to the file `vertica-informatica-plugin-server.jar`, you must also have the appropriate JDBC driver installed in the same directory (`/bin/javallib`). The JDBC driver you install must match your version of Vertica.

The following sample screen shows a typical Windows path for these files.



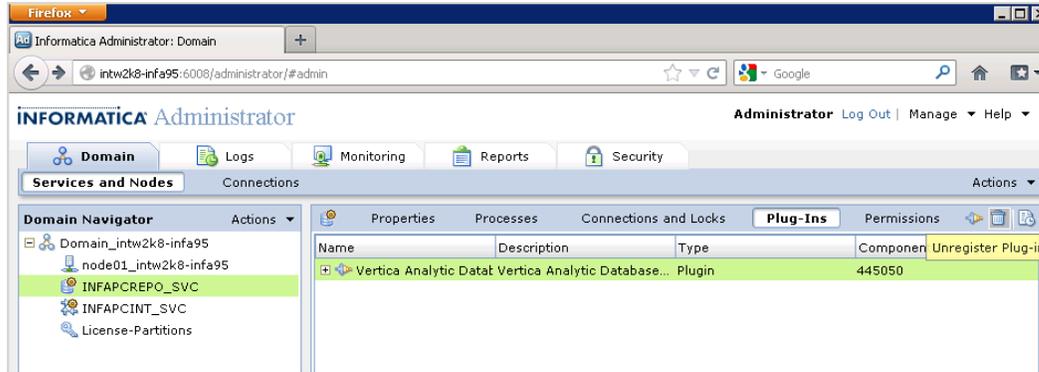
For all supported server operating systems (Linux, Solaris, AIX, HP-UX), you download and install the same file, `vertica-informatica-plugin-server.jar`.

Copy the file to the `server/bin/javallib` subdirectory of the directory where PowerCenter is installed.

Unregistering the Plug-in

Perform this procedure only if you need to unregister the plug-in.

1. Follow the procedure, Switch to Exclusive Mode.
2. Click the trashcan icon to unregister a plug-in.

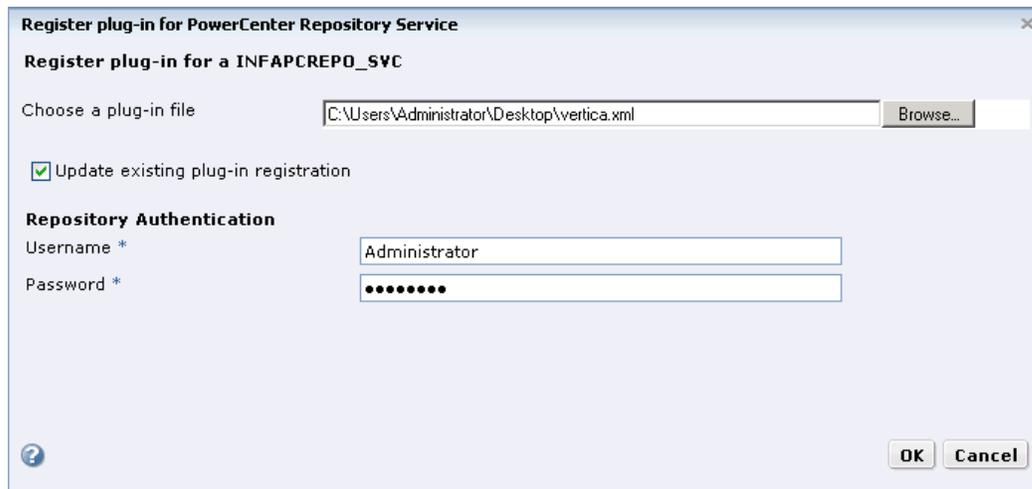


3. In the **Unregister plug-in** pop-up, enter your **Username** and **Password** to unregister the plug-in, and then click **OK**.
4. Follow the procedure, Switch Back to Normal Mode.

Updating the Plug-in

Follow this procedure only if you need to update the plug-in.

1. Follow the procedure, **Switch to Exclusive Mode**.
2. On the **Plug-ins** tab, click the icon for registering a plug-in.
3. On the **Choose a plug-in** field, click **Browse** and select the location of the `vertica.xml` file.
4. Check the box, **Update existing plug-in registration**.
5. Enter your **Username** and **Password** to unregister the plug-in, and then click **OK**.



6. Follow the procedure, **Switch Back to Normal Mode.**

Using the Vertica Plug-in with Informatica PowerCenter

Once you have installed the Vertica Plug-in for Informatica, you can use Vertica as a source or target in Informatica PowerCenter.

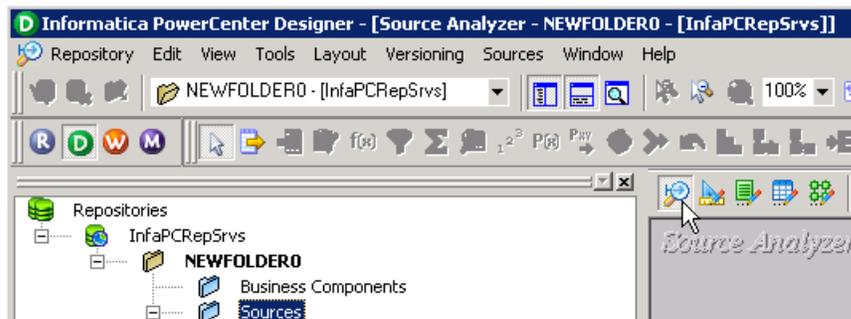
The simple examples in this section walk you through importing source and target, mapping, configuring, and starting your workflow.

Importing a Source Database Table

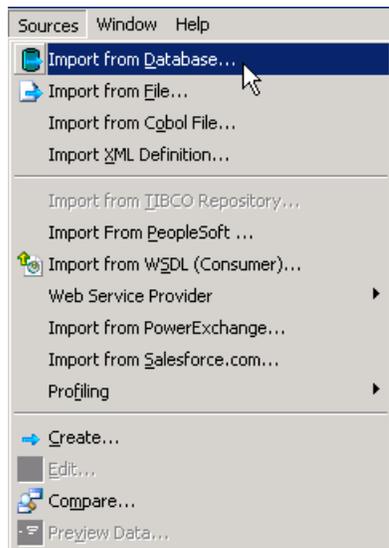
Note: Set up a DSN for your Vertica database before you perform the procedures that follow.

The following example shows how to import a source table from a Vertica database.

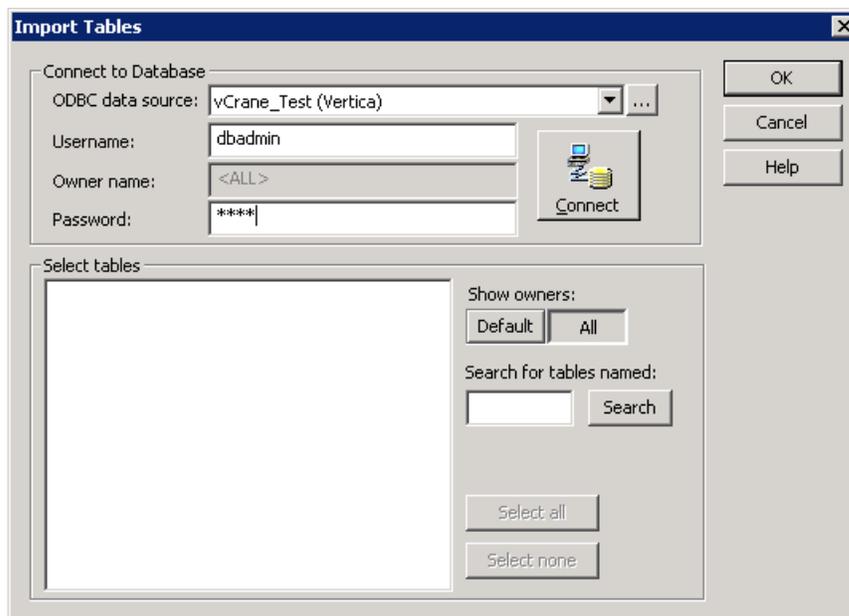
1. In Informatica PowerCenter Designer, select the folder in the repository where you want to create your Vertica source.



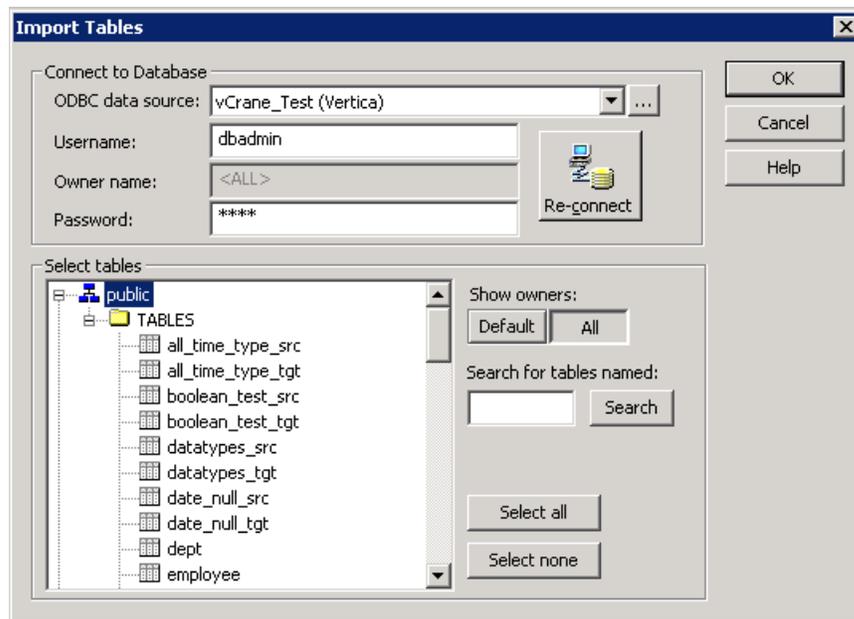
2. Click the Source Analyzer icon.
3. From the Sources list box, select **Import from Database**.



4. From the **Import Tables** dialog box, choose the name of your **ODBC** data source, and enter **Username** and **Password**.



5. Click **Connect**. Under **Select tables**, choose the schema **public**.



6. Choose a table and Click **OK**. For this example, choose only the table, **datatypes_src**. (You can choose a number of tables.)

The table appears in the Source Analyzer panel.

7. Change the table's database type to VERTICA. To do so, double-click the name of the table to launch the **Edit Tables** dialog box.
8. From the **Table** tab, **Database type** list box, choose **VERTICA**.

Note: As of Informatica Version 9.6.1 Hot Fix 2 and later, if you want to use the version of Vertica you downloaded and installed, select the Vertica option from the dropdown menu. If you intend to use the new Vertica Connector from Informatica, select VERTICA from the dropdown menu.

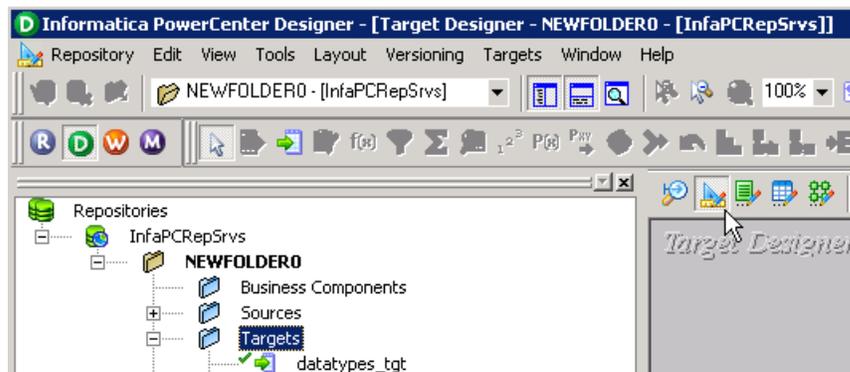
9. Click **OK**. You have imported a source table. Next, follow the procedure in, [Importing a Target Database Table](#).

Importing a Target Database Table

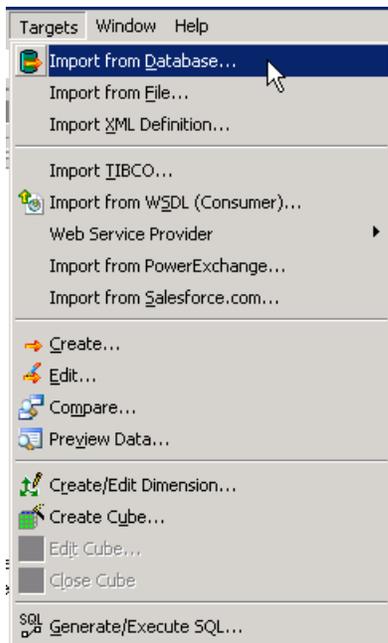
Note: Set up a DSN for your Vertica database before you perform the procedures that follow.

The following example shows how to import a target table from a Vertica database.

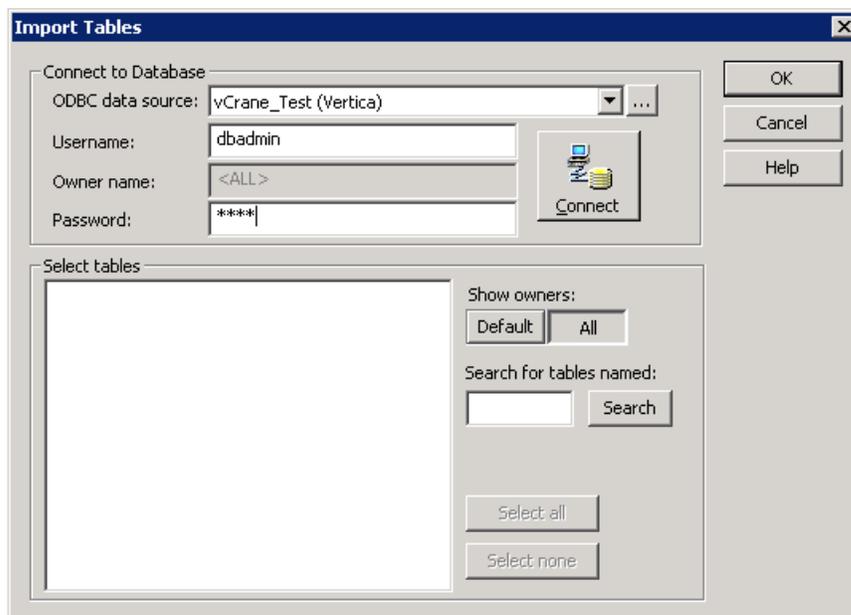
1. In Informatica PowerCenter Designer, select the folder in the repository where you want to create your Vertica target.



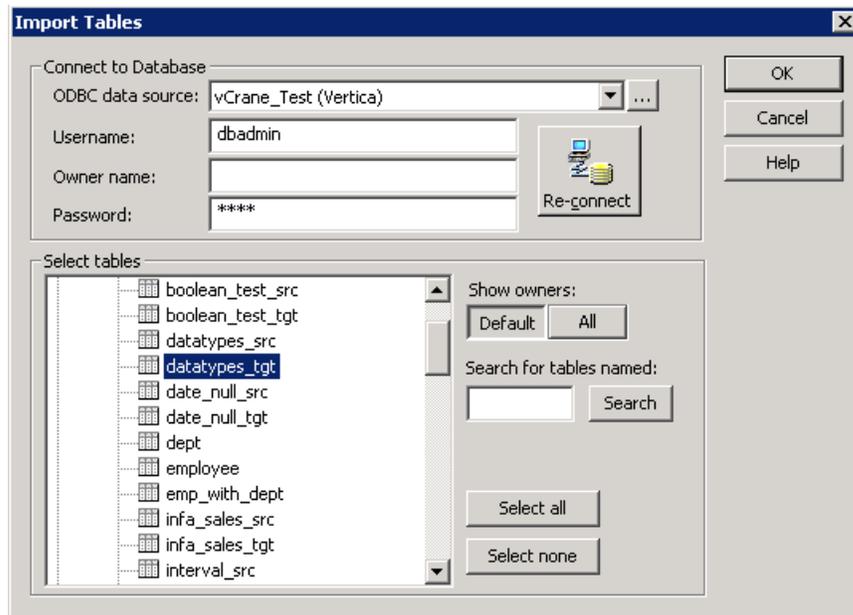
2. Click the Target Designer icon.
3. From the **Targets** list box, choose **Import from Database**.



4. From the **Import Tables** dialog box, choose the name of your **ODBC** data source, and enter **Username** and **Password**.



5. Click **Connect**. Under **Select tables**, choose the schema **public**.



6. Choose a table and Click **OK**. For this example, choose only the table, **datatypes_tgt**.

The table appears in the **Target Designer** panel.

7. Change the table's database type to VERTICA. Double-click the name of the table to launch the **Edit Tables** dialog box.
8. From the **Table** tab, **Database type** list box, choose **VERTICA**.

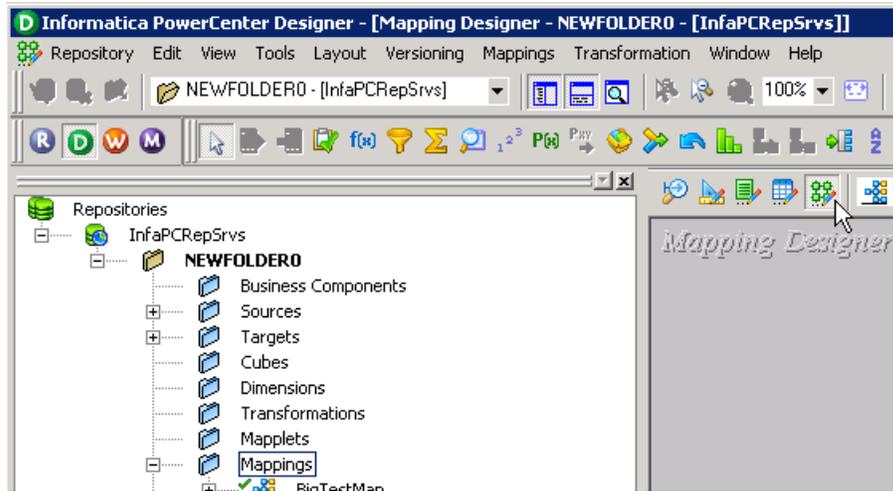
Note: As of Informatica Version 9.6.1 Hot Fix 2 and later, if you want to use the version of Vertica you downloaded and installed, select the Vertica option from the dropdown menu. If you intend to use the new Vertica Connector from Informatica, select VERTICA from the dropdown menu.

9. Click **OK**. You have imported a target table. Next, follow the procedure in, [Mapping Between Source and Target Tables](#).

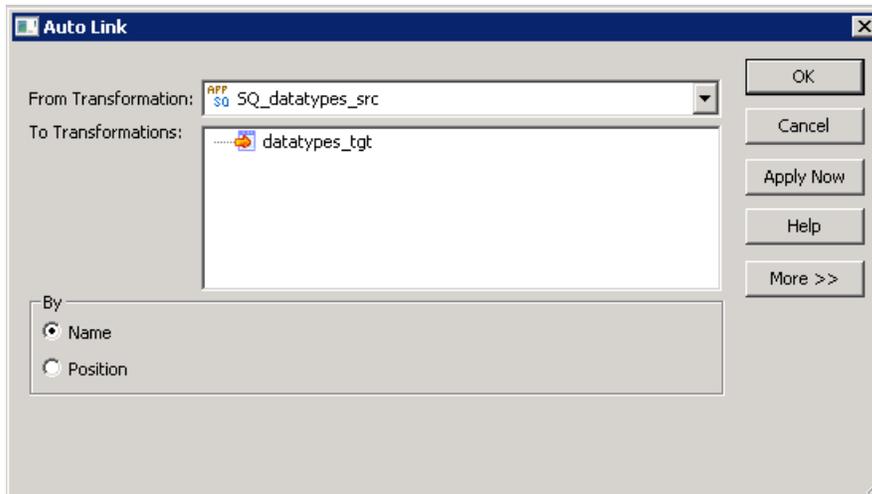
Mapping Between Source and Target Tables

Perform this procedure to create mapping between source and target tables.

1. In Informatica PowerCenter Designer, select the folder in the repository where you want to create your Vertica mapping.



2. Click the Mapping Designer icon.
3. From the **Mappings** list box, select **Create**.
4. Enter a mapping name and click **OK**.
5. Choose the source, **datatypes_src**, and drag it to the **Mapping Designer** window. An **Application Source Qualifier** also appears; your source is mapped to Informatica. (This example sets up a basic workflow and is not meant to be a realistic sample. Note also that this example shows that configuration changes would be required where both source and/or target are in Vertica databases.)
6. Drag your target, **datatypes_tgt**, to the **Mapping Designer** window.
7. From the **Layout** list box, select **Autolink by Name**.
8. Confirm the from and to transformations, and click **OK**.

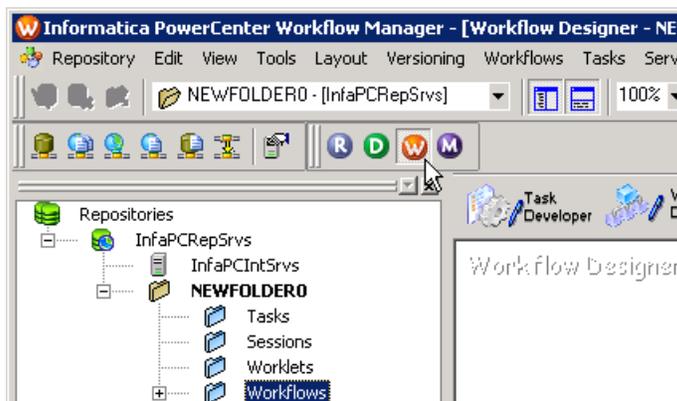


9. Save your work. Next, follow the procedure in, you create a workflow that uses your mapping. Follow the procedure in, [Creating a Workflow](#).

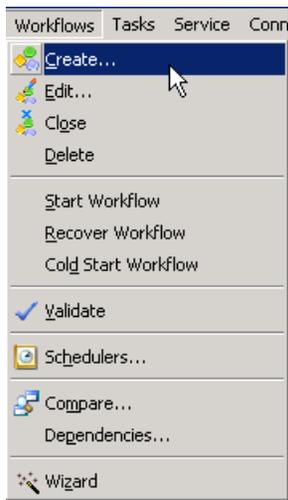
Creating a Workflow

Perform this procedure to create a workflow using the table mapping you previously created.

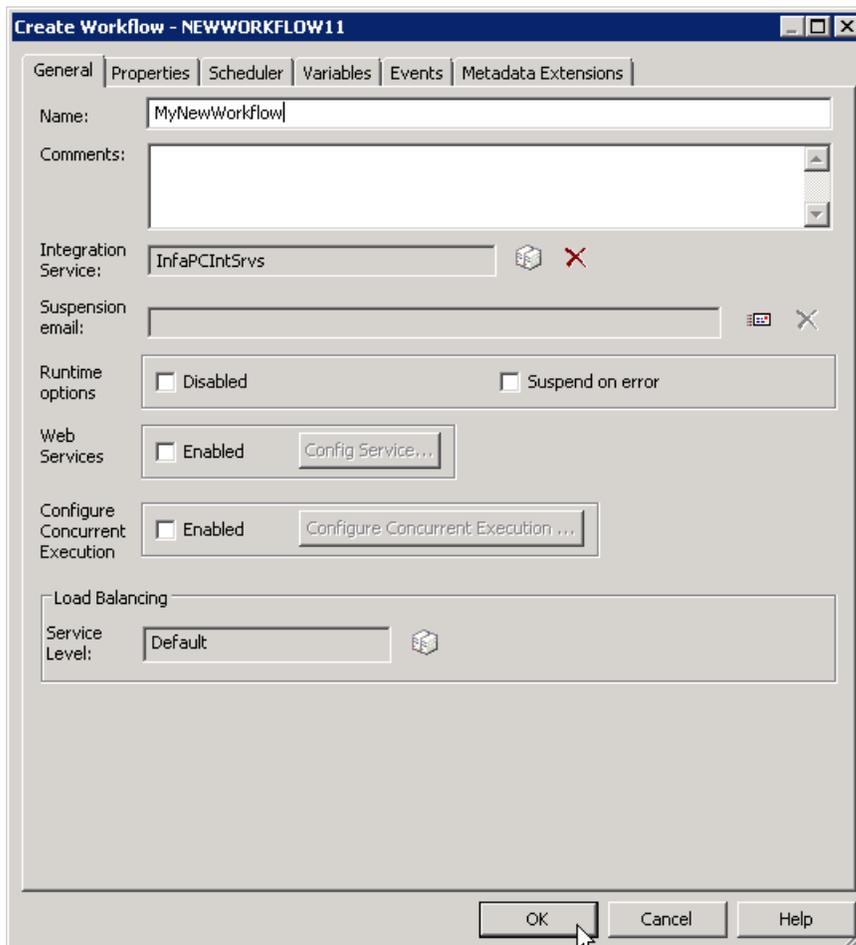
1. In Informatica PowerCenter, click the Workflow button to launch the Workflow Manager.



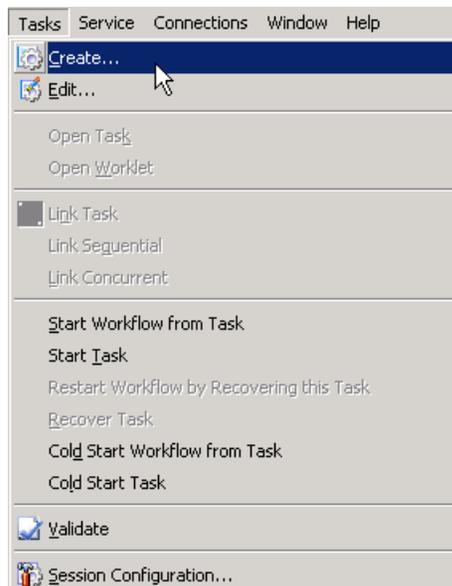
2. From the **Workflows** list box, select **Create**.



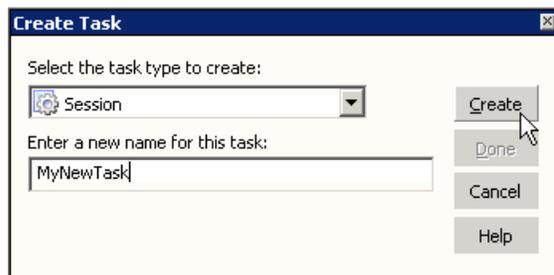
3. In the Create Workflow dialog box, enter a name for your new workflow and click **OK**.



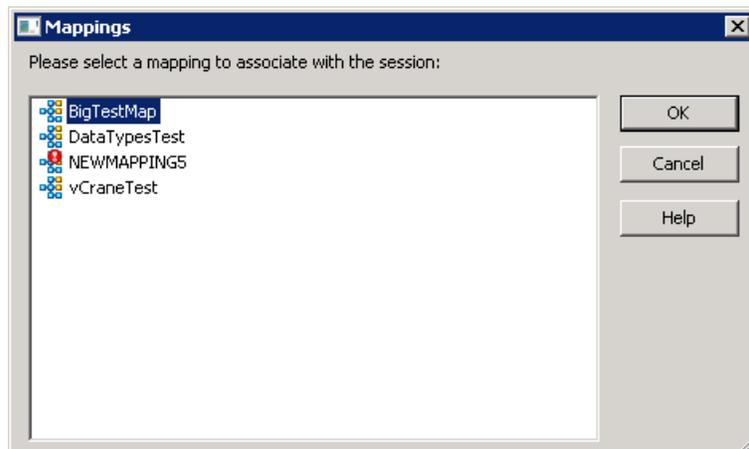
4. From the **Tasks** list box, select **Create**.



5. Enter a name for your new task, and click **Create**.

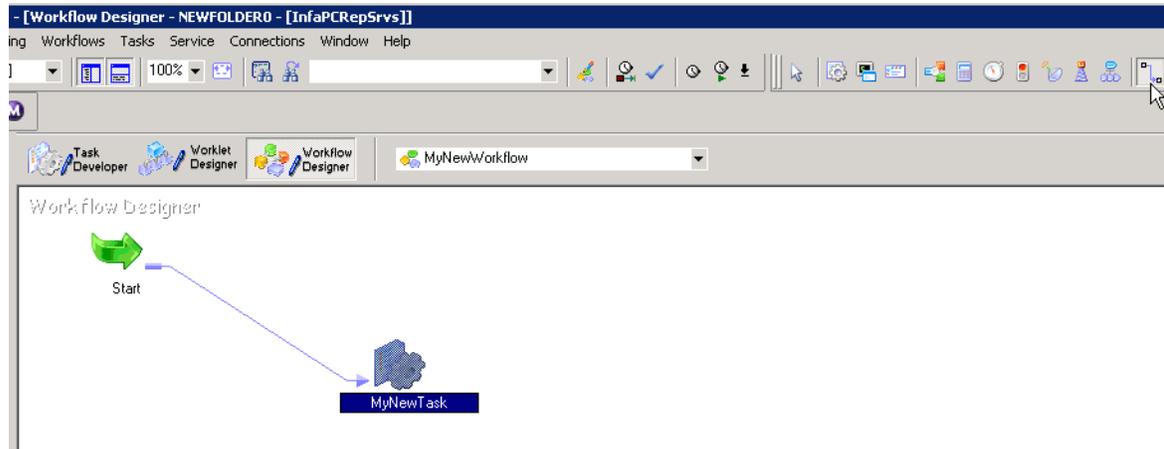


6. In the **Mappings** dialog box, choose the mapping to associate with the session and click **OK**.



7. Click **Done** on the Create Task dialog box.

8. In the Workflow Designer, drag your task to the right of **Start**.
9. Select the link tasks icon, and link **Start** to your task. Save your work.

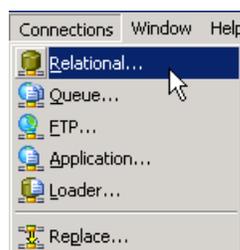


Next you configure your workflow connection to your database. Follow the procedure in, [Configuring Your Workflow Connections](#).

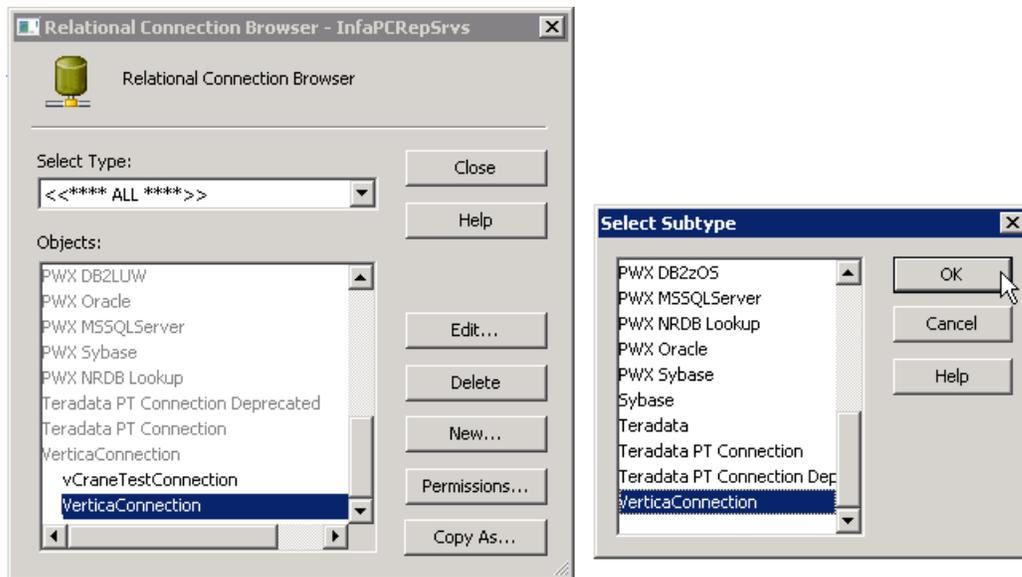
Configuring Your Workflow Connections

Perform this procedure to configure your workflow connection to your database.

1. While in Workflow Manager, from the **Connections** list box, select **Relational**.



2. From the **Relational Connection Browser** dialog box, select your Vertica connection object and click **New**.
3. From the **Select Subtype** dialog box, select **VerticaConnection** and click **OK**.



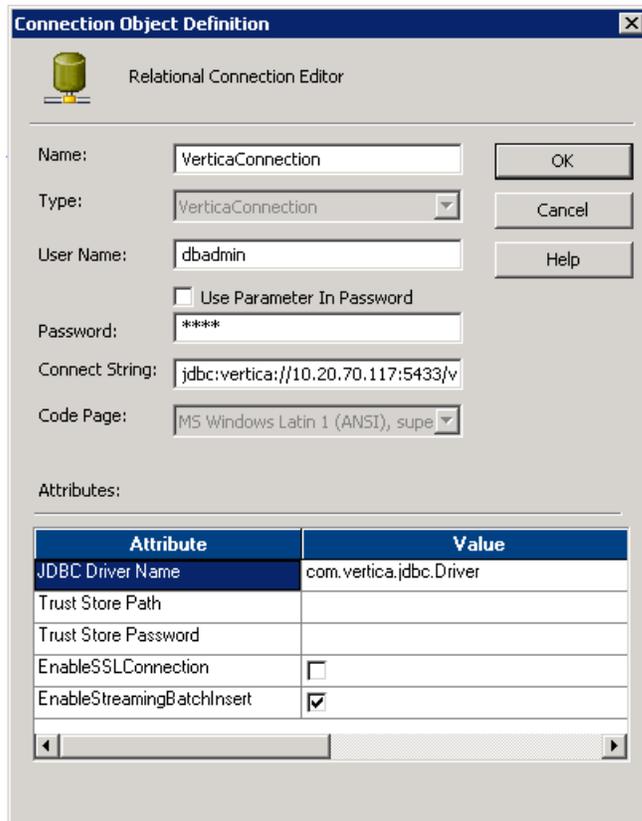
4. Fill in the details within the Connection Object Definition dialog box.
 - a. Fill in the **Name**, **User Name**, and **Password**.
 - b. Enter the connection string, which must have the format:
`jdbc:vertica://<ip>:<port>/<dbname>`
 - c. In the **JDBC Driver Name** field, enter **com.vertica.jdbc.Driver**
 - d. If you are connecting to a database that is running Vertica Release 8.1.x, select **EnableStreamingBatchInsert**.

Important: If you are running Vertica Release 7.x on either source or target database, enable (check off) **EnableStreamingBatchInsert**. Note that if you are running a previous version of Vertica you can still check **EnableStreamingBatchInsert**; previous versions will not experience the performance improvements, but setting the option has no detrimental impact.

- e. If your Vertica database is SSL enabled, check **EnableSSLConnection**, and enter a **Trust Store Path** and **Trust Store Password**.

Note: You must enter a complete path for Trust Store Path (e.g., C:\Users\Administrator\Desktop\verticastore).

- f. Click **OK**.

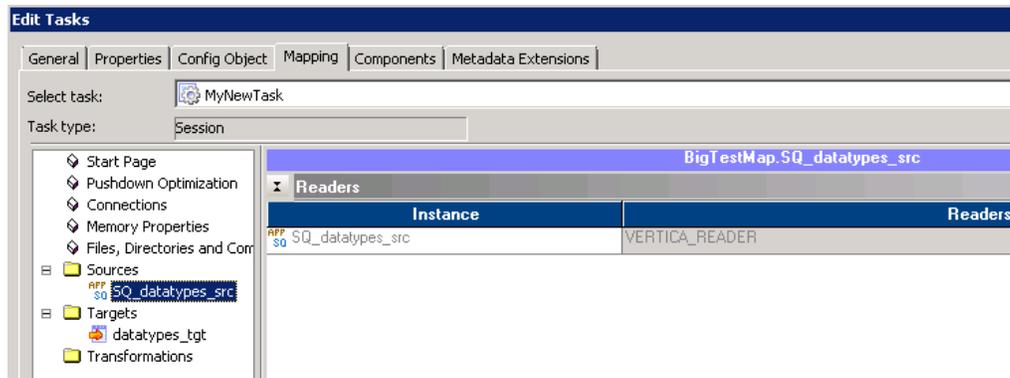


Next you configure your source and target for the workflow. Follow the procedure in, [Configuring Source and Target and Starting Your Workflow](#).

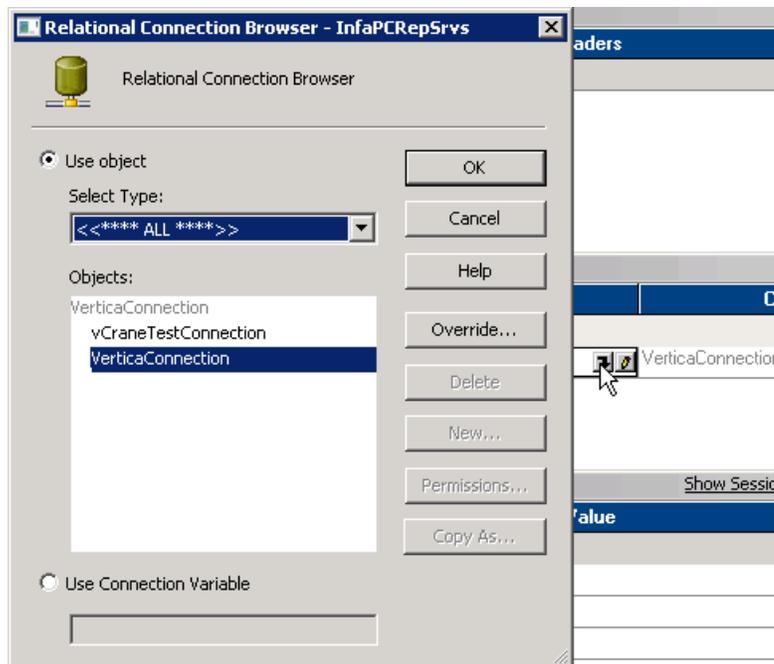
Configuring Source and Target and Starting Your Workflow

Perform this procedure to configure source and target for your workflow. (This procedure assumes all previous procedures have been completed.)

1. While in Workflow Designer, double-click your task to launch the **Edit Tasks** dialog box.
2. Click the **Mapping** tab.
3. Configure the source.
 - a. Under the **Sources** folder, select your source. The **VERTICA_READER** is listed under **Readers**.



b. Click the down arrow icon to bring up the **Relational Connection Browser** dialog box.



c. Choose your object and click **OK**.

4. Configure the target.

a. Under the **Targets** folder, select your target. The **VERTICA_WRITER** is listed under **Writers**.

b. Click the down arrow icon to bring up the **Relational Connection Browser** dialog.

c. Choose your object and click **OK**. Save your work.

5. While in Workflow Manager, from the Workflows list box, select **Start Workflow**. (The Workflow Monitor opens.)

Accessing and Setting Plug-in Features

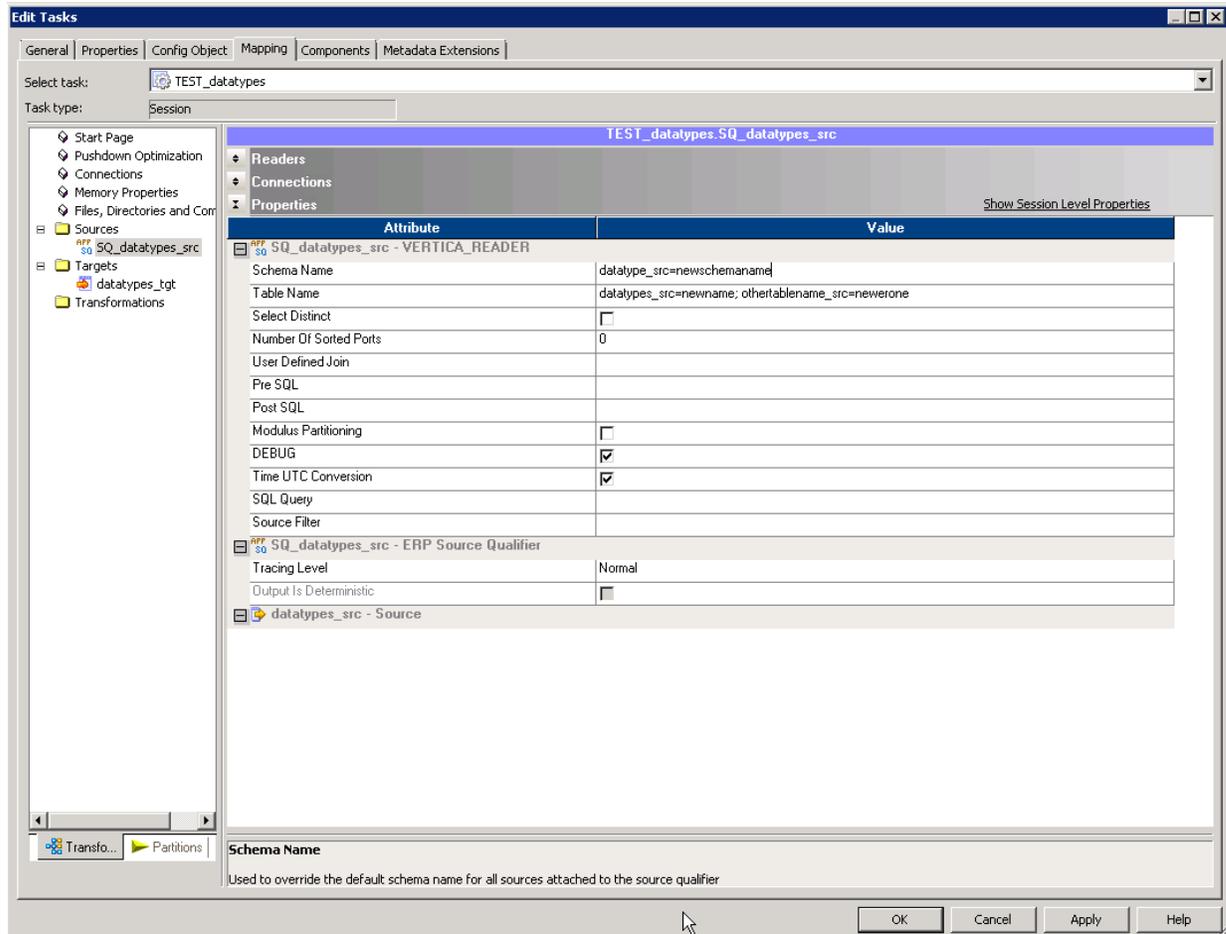
You can access the Vertica plug-in features through any workflow task. This section discusses how to access the features available through the Vertica plug-in.

Accessing VERTICA_READER Plug-in Attributes

Note that, when you use Vertica as source/reader, the plug-in supports only pass-through partitioning. When you use Vertica as target/writer, the plug-in supports pass-through partitioning and key range partitioning. (Note that the key value partitioning source should not be through the Vertica plug-in.)

1. While in Workflow Manager, double-click any task.
2. In the **Edit Tasks** dialog, select the Mappings tab.
3. Select your source qualifier (this example uses `SQ_datatypes_src`).
4. Minimize the **Readers** and **Connections** areas to focus on **Properties**.

Plug-in features are listed under the Properties area.



The table that follows lists and describes the plug-in attributes for source/reader.

Attribute	Value
Schema Name	<p>Schema Name allows you to override the default schema name of the mapped table. If you have only one source, you can override the schema name by simply entering the new schema name.</p> <p>Where you have many sources in a mapping that all go to the one source qualifier, you override schema names with the following format. Use a semicolon as separator:</p> <p><code><old_table_name>=<new_schema_name>; <another_table_name>=<another_new_schema_name></code></p> <p>Important: In the format given above, you do not actually enter the schema name to change the schema name. Instead, you enter the table</p>

Attribute	Value
	<p>name in the schema name field (to the left of the equal sign). You then provide the new schema name as given in the format above (to the right of the equal sign; the equal sign acts to set the new name). Use a semicolon as separator for multiple schema name changes.</p> <p>Example:</p> <pre>datatype_src=newschemaname</pre>
Table Name	<p>Table Name changes are similar to Schema Name changes in regards to how you format the changes. If you have only one table name to change, you just enter the new table name.</p> <p>If you have more than one table name to change, you use the following format:</p> <pre><old_table_name>=<new_table_name>; <another_table_name>=<another_new_table_name></pre> <p>Example:</p> <pre>datatypes_src=newname;othertablename_src=newerone</pre> <p>Important: If using the Schema Name option along with the Table Name option, note that the Schema Name entry uses the old table name versus the replacement name you have added here in the Table Name option. An example follows.</p> <p>Schema Name entry:</p> <ul style="list-style-type: none"> • <code>tbl1=new-schema-name</code> <p>Table Name entry:</p> <ul style="list-style-type: none"> • <code>tbl1=tbl2</code> <p>That is, you would not specify <code>tbl2</code> under the Schema Name entry.</p>
Select Distinct	If selected, returns only distinct values.
Number of Sorted Ports	Sorts incoming data, specifying order by ports.
User Defined	Allows you to do overrides to your mapping, enabling you to do very specific

Attribute	Value
Join	<p>joins (e.g., inner and outer joins). When you click the arrow to the right of the field, a SQL box pops up. What you enter in the box becomes a custom join clause added to the generated SQL statement.</p> <p>Basically you can enter a typical join statement with SELECT being assumed.</p> <p>Examples:</p> <ul style="list-style-type: none"> • <code>s1.a.col = s2.b.col</code> (where <code>s1</code> and <code>s2</code> are schema names) • <code>a JOIN b ON a.col = b.col</code> • <code>a.col = b.col</code>
Pre SQL	<p>Enter complete SQL statements that run before you read a table.</p> <p>For example, truncate or add data to a table before you read it.</p>
Post SQL	<p>Enter complete SQL statements that run after you read a table.</p>
Modulus Partitioning	<p>If selected, performs modulus partitioning (no replication of data) rather than straight pass-through partitioning.</p> <p>Check this option only if using pass-through partitioning.</p>
Time UTC Conversion	<p>If selected, keeps time synchronized when you are using Vertica as both source and target. Time data changes to UTC time zone.</p> <p>Select this option and the writer option only when using Vertica as both source and target.</p> <p>If unchecked, when you are using Vertica as both source and target, time data changes to the JVM time zone.</p>
SQL Query	<p>Overrides the entire query rather than overriding just a portion of a query.</p>
Source Filter	<p>Allows you to do overrides to your mapping by overriding the WHERE clause (similar to the way User Defined Join allows you to override a join clause).</p>

Accessing VERTICA_WRITER Plug-in Attributes

1. While in Workflow Manager, double-click any task.
2. In the **Edit Tasks** dialog box, select the Mappings tab.
3. Select your source qualifier (this example uses `datatypes_tgt`).
4. Minimize the **Readers** and **Connections** areas to focus on **Properties**.

Plug-in features are listed under the **Properties** area.

The screenshot shows the 'Edit Tasks' dialog box with the 'Properties' tab selected. The task name is 'copylocalexceptiofile.tchar_tgt'. The 'Properties' section is expanded to show a table of attributes for the 'VERTICA_WRITER' plug-in. The table has two columns: 'Attribute' and 'Value'.

Attribute	Value
tchar_tgt - VERTICA_WRITER	
Schema Name	
Truncate target table option	<input checked="" type="checkbox"/>
Pre SQL	
Post SQL	
Target Table Name	
Copy Direct	<input type="checkbox"/>
Reject file directory	C:\
Reject filename	reject.txt
Insert	<input checked="" type="checkbox"/>
Delete	<input type="checkbox"/>
Update	None
Time UTC Conversion	<input type="checkbox"/>
Copy Local Method	Delimited
DEBUG	<input checked="" type="checkbox"/>
Target Number	0
tchar_tgt - Target	
Partial Load Recovery	None

The table that follows lists and describes the plug-in attributes for target/writer.

Attribute	Value
Schema Name	Change the schema name by entering a new name in this field.
Truncate Target Table	Select this option if you have a workflow that should truncate its targeted table before loading data.
Pre SQL	Enter complete SQL statements that run before you write to a table.
Post SQL	Enter complete SQL statements that run after you write to a table.
Target Table Name	Change the target table name by entering a new name in this field.
Copy Direct	When selected, writes directly to the ROS container. More efficient for bulk loading.
Reject file directory	Reject file directory and Reject file name work in tandem to record rejected rows. In this field you specify the directory path for the reject file.
Reject file name	Specify the name of the file that holds the rejected rows. There is one log file per partition. If there are multiple log files, a number is appended to the file names. For example, rejects.txt would become rejects_01.txt and rejects_02.txt.
Insert	If selected, makes insert the update strategy for the target.
Delete	If selected, makes delete the update strategy for the target.
Update	List box offers standard update options for target.
Time UTC Conversion	If selected, keeps time synchronized when you are using Vertica as both source and target. Time data changes to UTC time zone. Select this option and the reader option only when using Vertica as both source and target. If unchecked, when you are using Vertica as both source and target, time data changes to the JVM time zone.
Copy Local Method	Choose method from list box. None. The default; in this case, no local copy method is used to stream data. Important: To take advantage of increased performance of

Attribute	Value
	<p>EnableStreamingBatchInsert, you must set Copy Local Method to None.</p> <p>Delimited. Creates a stream with pipes and newline delimiters. This method fails if there are pipes or newline delimiters in the data you are transmitting.</p> <p>Native Varchar Converts to new format. Use if data is mostly some form of strings.</p> <p>Native binary Serializes java objects to Vertica objects.</p>
Target Number	<p>Default behavior. The Target Number default is zero. If you leave the default (or set Target Number to any number less than or equal to 1), the plug-in uses the IP address you specify in the Connection String when configuring workflow connections. See Configuring Your Workflow Connections for more information on where you specify the IP address.</p> <p>In regards to load balancing, the Target Number default operates as follows.</p> <ul style="list-style-type: none">• If the IP Address you specified in the Connection String is not an Vertica node, the plug-in targets that IP Address. Any load balancing policy enabled on the IP Address is used.• If the node you specified in the Connection String is an Vertica node, note the following: Vertica native connection load balancing is off by default; in this case, the plug-in uses the IP address you specify in the connection string. If Vertica load balancing is enabled, then the plug-in targets nodes as defined by the load balancing scheme on the Vertica node. <p>Non-default behavior. You override the plug-in's default behavior by setting the option Target Number to a value greater than 1 (up to 16). When the value of the Target Number is set to greater than 1, the plug-in targets that number of Vertica nodes and uses ROUNDROBIN as its load balancing scheme. This overrides the native Vertica load balancing scheme.</p> <p>For information on native connection load balancing, refer to, About About Native Connection Load Balancing in the Administrator's Guide. For information on setting a load balancing policy on a Vertica server, refer to SET_LOAD_BALANCE_POLICY in the SQL Reference Manual.</p>

Setting EnableStreamingBatchInsert

If you are connecting to a database that is running Vertica Release 8.1.x, for best performance, you should always implement **EnableStreamingBatchInsert**.

If you are running an earlier version of Vertica, setting **EnableStreamingBatchInsert** has no impact on performance.

Find the full procedure for accessing the setting in the section, [Configuring Your Workflow Connections](#). For a workflow that is already set up:

1. While in Workflow Manager, select a task.
2. From the **Connections** list box, select **Relational**.
3. Click **Edit**.
4. In the **Connection Object Definition** dialog box, check off **EnableStreamingBatchInsert**.
5. Click **OK**. Save your work.

Important: To take advantage of increased performance of **EnableStreamingBatchInsert**, you must set Copy Local Method to **None**.

Enabling SSL

If your Vertica database is SSL enabled, then within the **Connection Object Definition** dialog, check **EnableSSLConnection**, and enter a **Trust Store Path** and **Trust Store Password**.

Note: You must enter a complete path for **Trust Store Path** (e.g., C:\Users\Administrator\Desktop\verticastore).

For a simple example of where to access the SSL setting, refer to [Configuring Your Workflow Connections](#).

Best Practices

This section includes memory requirement considerations and other tips.

Setting Memory Properties for a Task

Micro Focus recommends that you increase memory allocation to improve performance.

Note: The default buffer size for Informatica PowerCenter is set very conservatively. These settings can cause PowerCenter to send Vertica many small batches, rather than a few large batches. The overhead of these many small batches can cause loading performance issues. To resolve these performance issues, you should change PowerCenter's batch size settings. Your specific settings depend upon your system resources and needs.

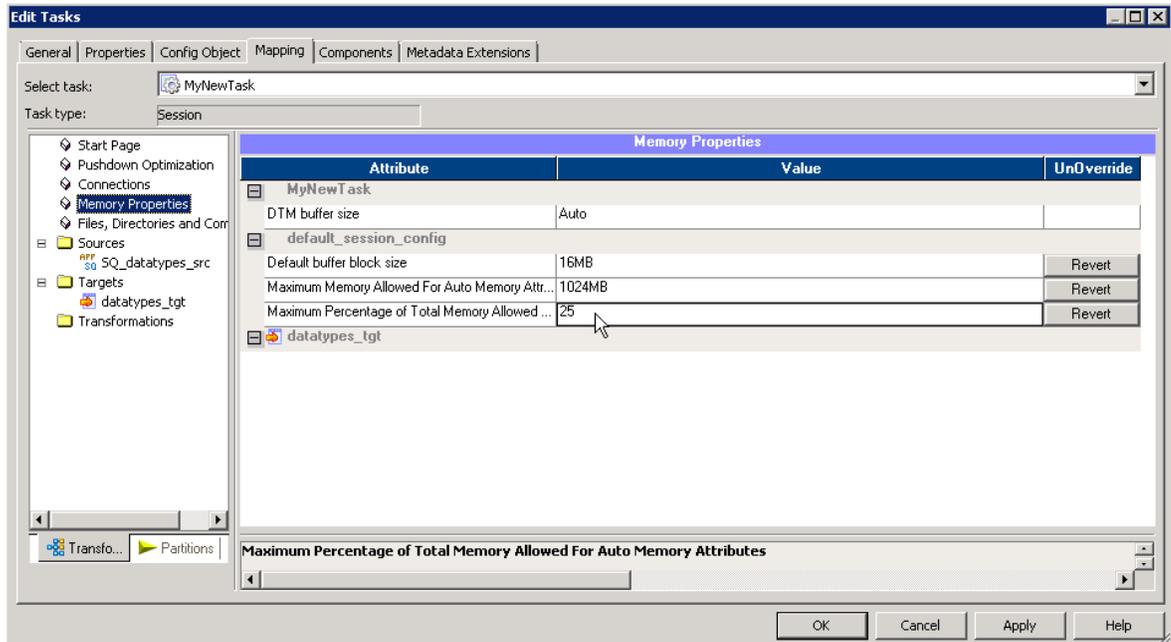
Perform the following procedure to increase memory allocation for a task.

1. While in Workflow Manager, double-click the task that connects to Vertica.
2. In the **Edit Tasks** dialog box, under the **Mapping** tab, select **Memory Properties**. Considering your task requirements and your system limitations, set the following attributes.

Note: Allocate more memory than mentioned here according to your system limitations and needs. The settings given may not be realistic for the tasks you intend to perform.

- a. Set **Default buffer block size** to at least 16 MB.
- b. Set **Maximum Memory Allowed for Auto Memory Attributes** to at least 512 MB.
- c. Set **Maximum Percentage of Total Memory Allowed for Auto Memory Attributes** to

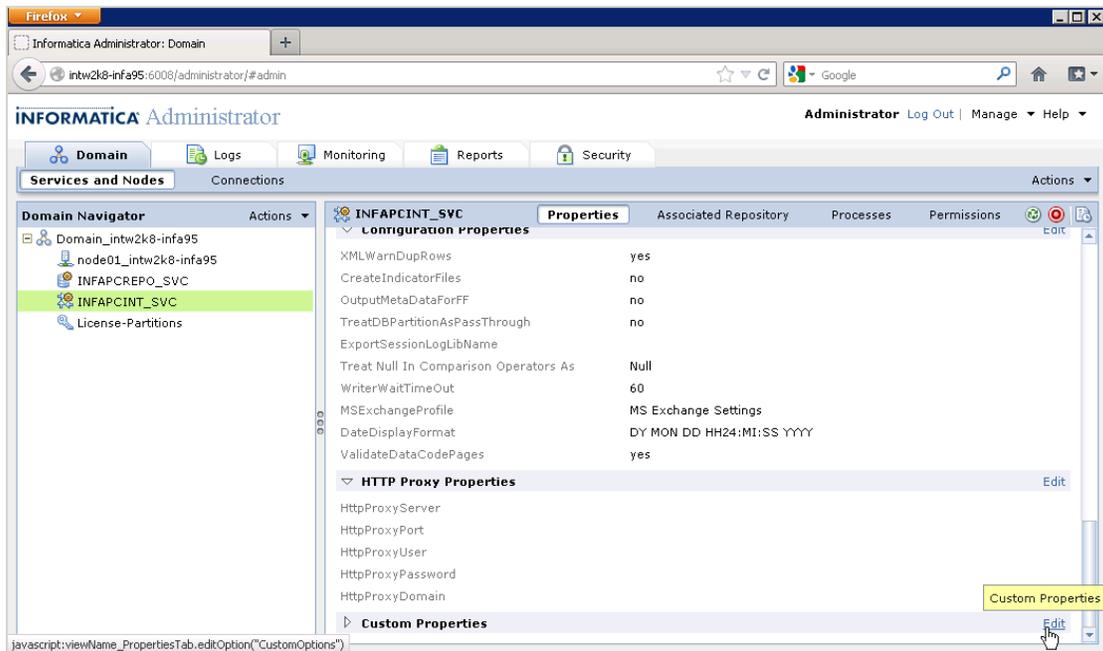
at least 25.



Setting JVM Memory Properties

Micro Focus recommends the following settings for your JVM (Java Virtual Machine).

1. Log on to the PowerCenter domain's Administration Console.
2. Select the **Domain** tab, and click **Services and Nodes**.
3. In the **Domain Navigator**, click the entry for the PowerCenter Integration Service.
4. Click **Edit** next to the **Custom Properties** section.

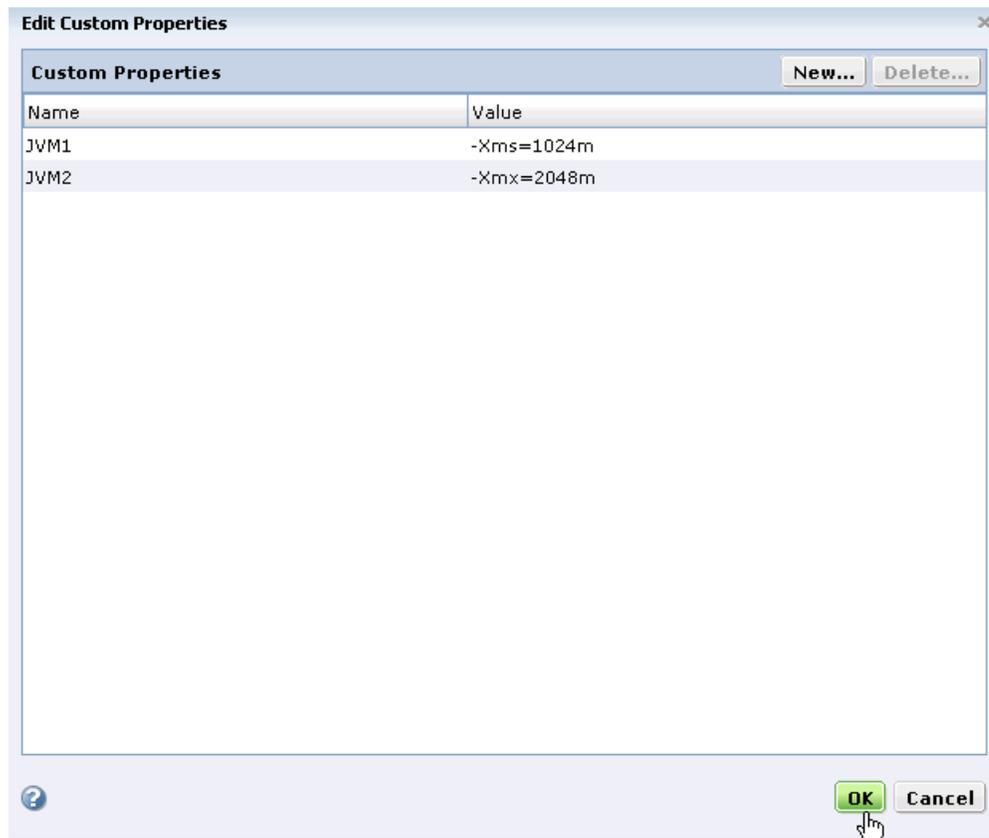


5. In the **Edit Custom Properties** dialog box, click **New**.
 - a. Enter a name.
 - b. Set a minimum heap memory size to at least 1024 m. Enter `-Xms=1024m`
 - c. Click **OK**.

6. In the **Edit Custom Properties** dialog box, click **New** again.
 - a. Enter a name.
 - b. Set a maximum heap memory size to at least double the minimum you just entered.

Enter `-Xmx=2048m`

c. Click **OK**.



Communicating when Informatica and Vertica Are on Different Networks

This best practice concerns communication with a Vertica cluster from Informatica when Informatica and Vertica are on separate networks. If Informatica and Vertica are on the same network, you do not need to implement the changes described here.

If Informatica and Vertica are on separate networks:

- Set up a public network for import/export and specify an export address for your individual nodes. See [Using Public and Private IP Networks](#) in the Administrator's Guide, specifically the section, [Identify the Database or Nodes Used for Import/Export](#). See also [ALTER DATABASE](#) and [ALTER NODE](#) in the SQL Reference manual for subnet and node-related tasks.

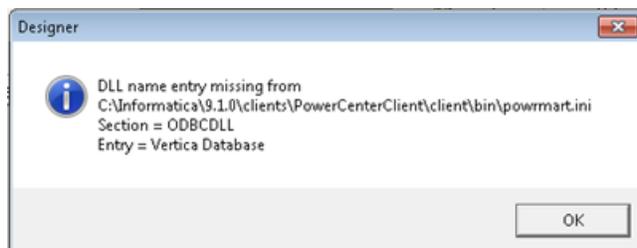
The Vertica plug-in for Informatica accesses the EXPORT_ADDRESS column of the V_Catalog schema. (For information on viewing the EXPORT_ADDRESS column, see the SQL Reference manual, specifically the section [NODES](#) in the [V_Catalog Schema](#).)

- Once the public network is set-up properly, and an export address is assigned to each node, Informatica can then read and write to a Vertica cluster on a different network.

Important: If you are using the Vertica plug-in for Informatica on Vertica Release 6.x, the Informatica user must have the PSEUDOSUPERUSER role to access the export addresses on the different network. For general information on the PSEUDOSUPERUSER role, see the Administrator's Guide, [PSEUDOSUPERUSER Role](#). If you are using Vertica Release 7.0.x, the Informatica user does not need the PSEUDOSUPERUSER role.

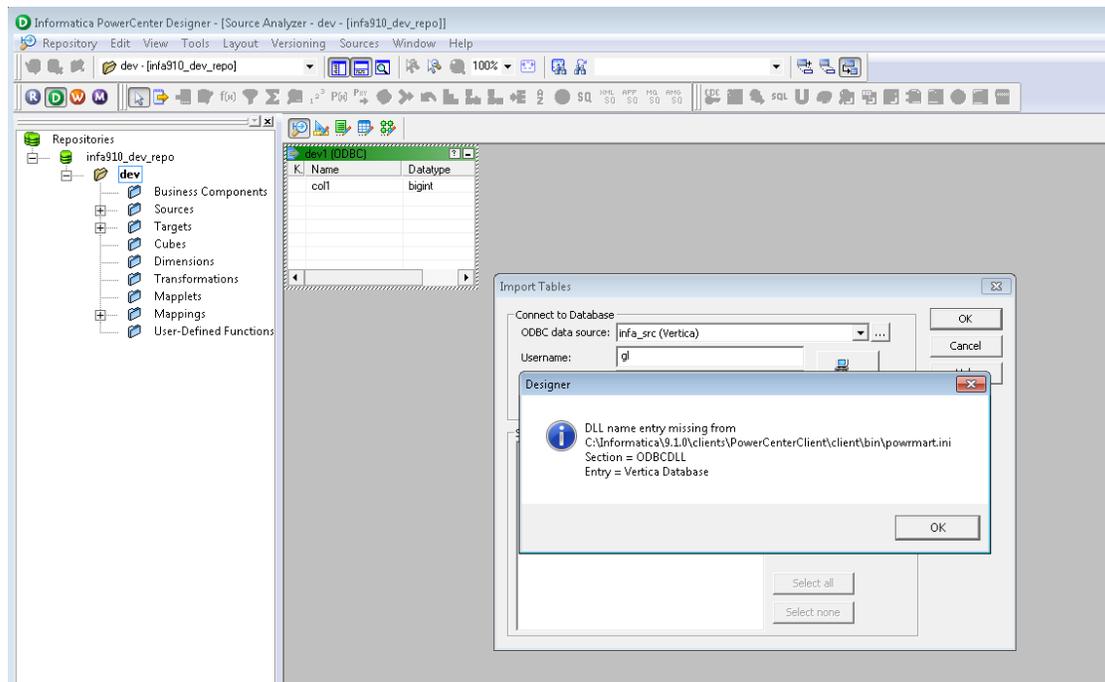
Modifying powrmart.ini

When reading or importing a table, you can receive a pop-up warning, such as the following, concerning a missing DLL.

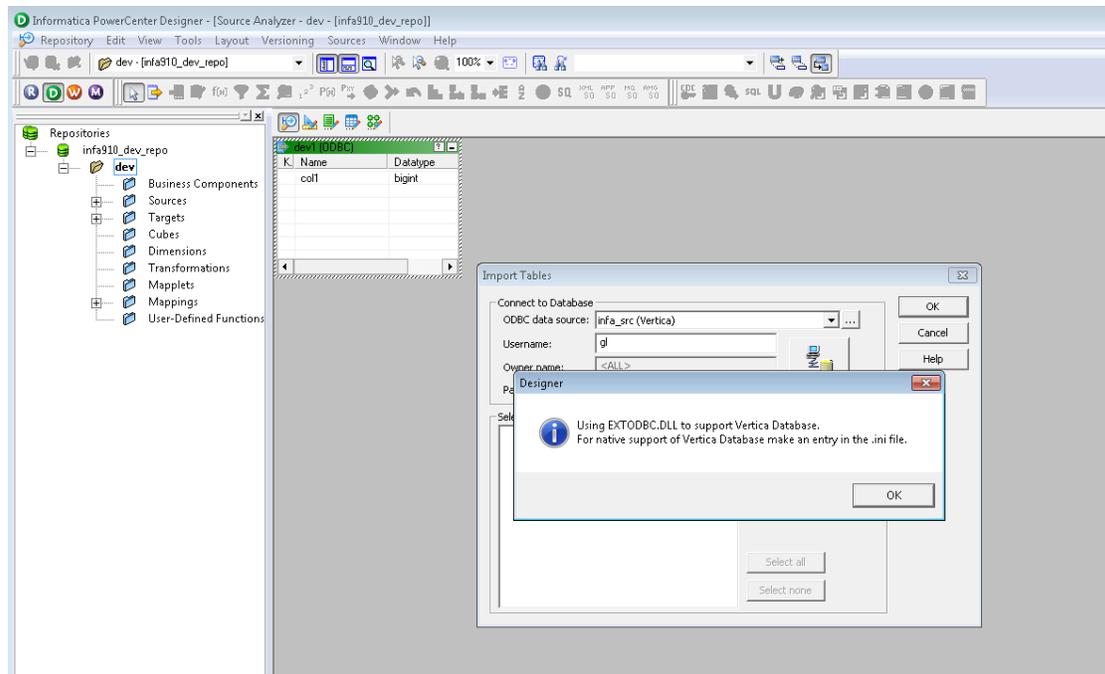


You can make a correction so that the warning no longer appears by adding the line `Vertica Database=PMODBC.DLL` to the `powrmart.ini` file in the section `ODBCDLL`. See the following example.

1. Using the Import Tables option in the Informatica PowerCenter Designer, the system displays a pop-up warning about a missing DLL. Click **OK**.



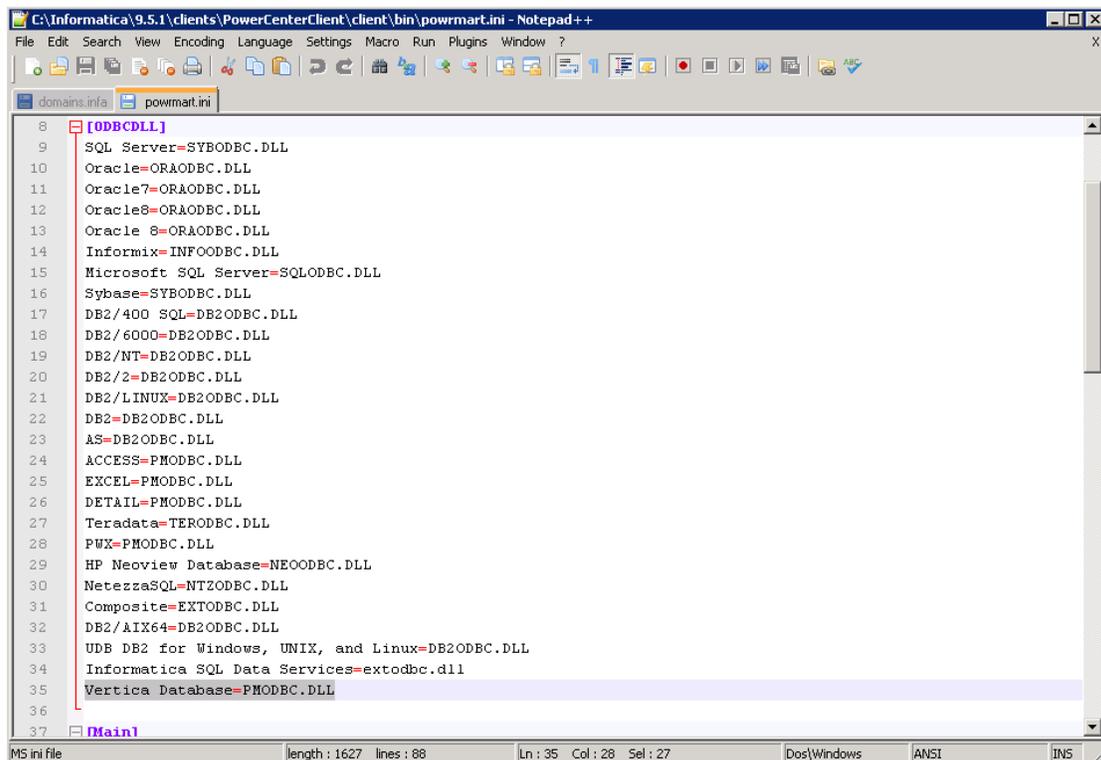
2. The system displays another pop-up letting you know that Informatica is using EXTODBC.DLL to support Vertica. Click OK.



Note: The two pop-up messages are warnings only and do not affect the import of tables or the execution of workflows.

3. Open the `powrmt.ini` file for editing.
4. Add the following line in the section `ODBCDLL`.

Vertica Database=PMODBC.DLL

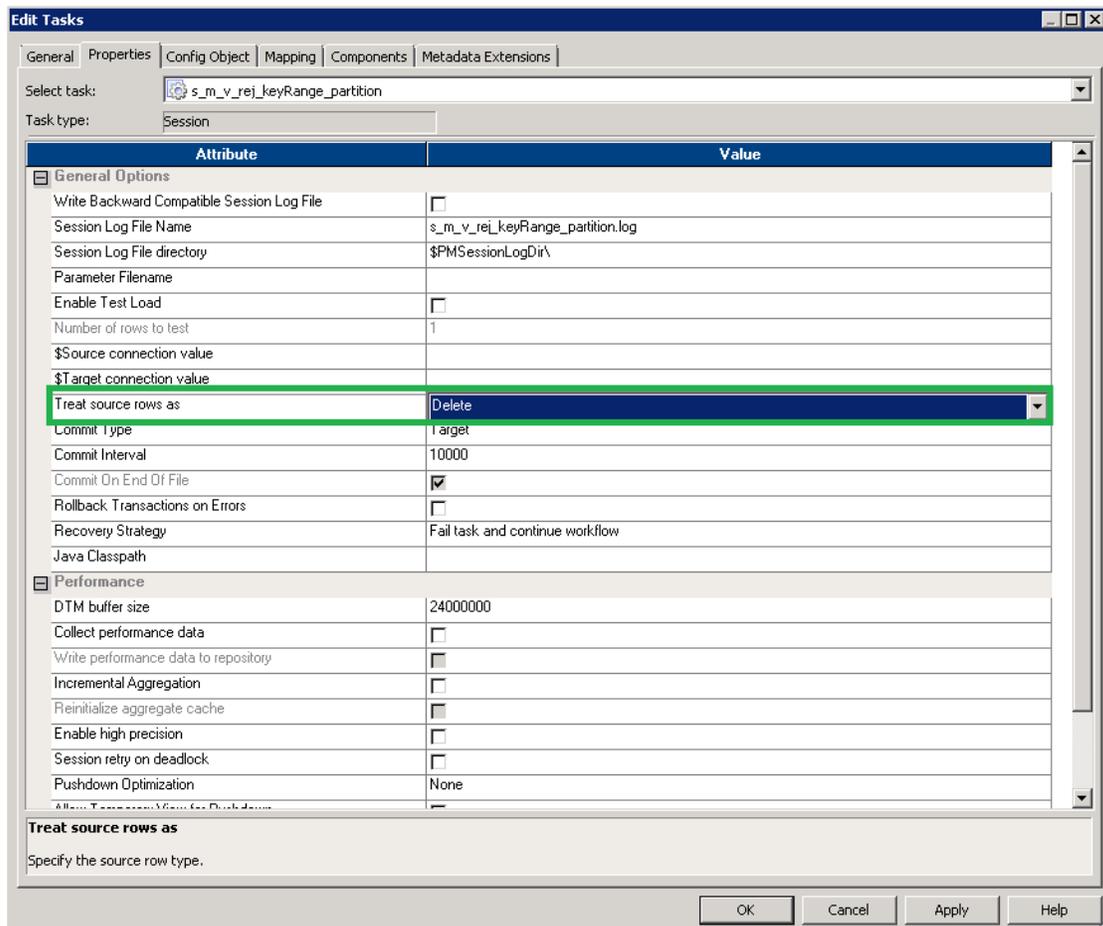


```
C:\Informatica\9.5.1\clients\PowerCenterClient\client\bin\powrmt.ini - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
domains.infa powrmt.ini
8 [ODBCDLL]
9 SQL Server=SYBODBC.DLL
10 Oracle=ORAODBC.DLL
11 Oracle7=ORAODBC.DLL
12 Oracle8=ORAODBC.DLL
13 Oracle 8=ORAODBC.DLL
14 Informix=INFOODBC.DLL
15 Microsoft SQL Server=SQLODBC.DLL
16 Sybase=SYBODBC.DLL
17 DB2/400 SQL=DB2ODBC.DLL
18 DB2/6000=DB2ODBC.DLL
19 DB2/NT=DB2ODBC.DLL
20 DB2/2=DB2ODBC.DLL
21 DB2/LINUX=DB2ODBC.DLL
22 DB2=DB2ODBC.DLL
23 AS=DB2ODBC.DLL
24 ACCESS=PMODBC.DLL
25 EXCEL=PMODBC.DLL
26 DETAIL=PMODBC.DLL
27 Teradata=TERODBC.DLL
28 PWX=PMODBC.DLL
29 HP Neoview Database=NEOODBC.DLL
30 NetezzaSQL=NTZODBC.DLL
31 Composite=EXTODBC.DLL
32 DB2/AIX64=DB2ODBC.DLL
33 UDB DB2 for Windows, UNIX, and Linux=DB2ODBC.DLL
34 Informatica SQL Data Services=extodbc.dll
35 Vertica Database=PMODBC.DLL
36
37 [Main]
MS ini file length : 1627 lines : 88 Ln : 35 Col : 28 Sel : 27 Dos\Windows ANSI INS
```

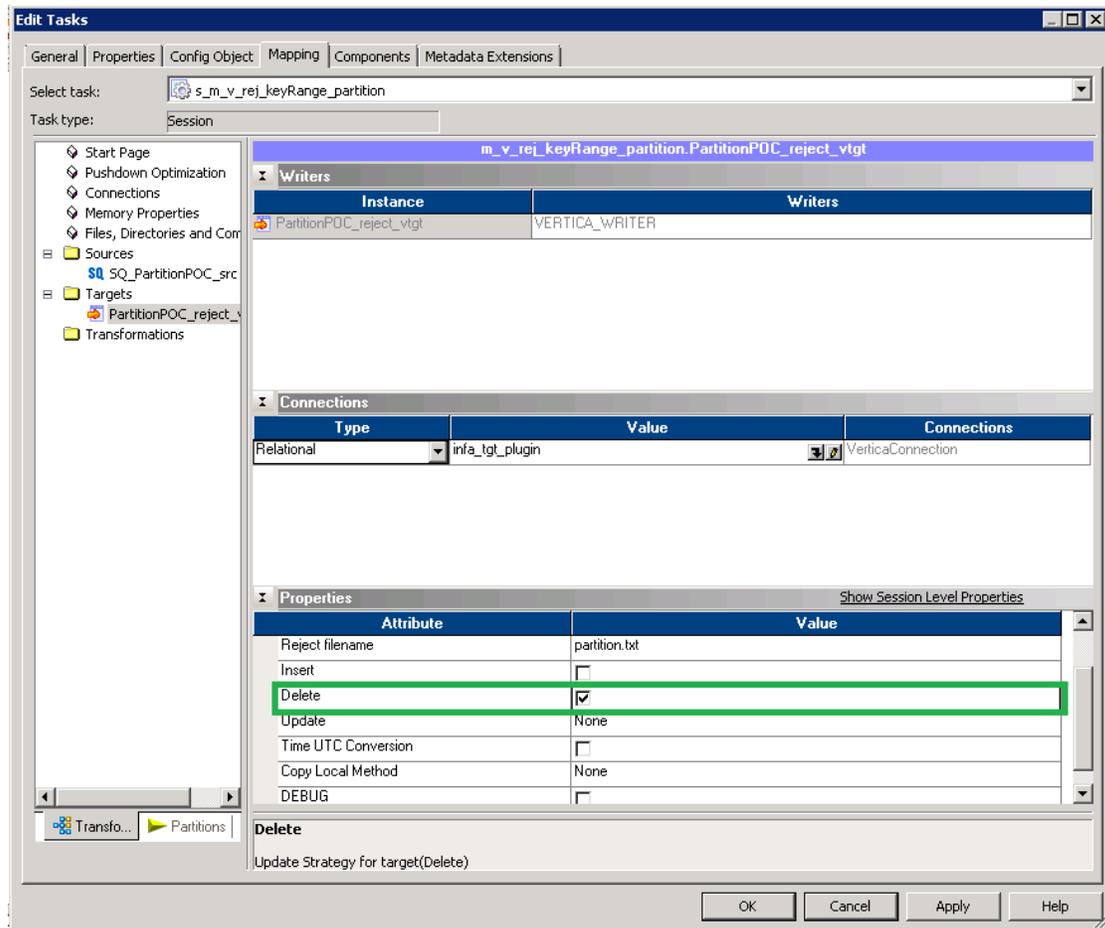
Deleting Records on a Target Table

Follow these notes to delete records on a target table.

1. On the target table, you must have defined a primary key. From **Edit Tasks**, select the **Properties** tab, and, under **General Options**, set **Treat Source row as** to **Delete**.



2. Also on the target table, from the **Edit Tasks** dialog, choose the **Mapping** tab. Under **Properties**, check-off **Delete**.



Vertica Error Messages

Welcome to the Vertica Error Codes guide. This guide is mainly for developers who need to understand how Vertica's error codes relate to error states returned by the ODBC and JDBC drivers.

The Different Ways Vertica Reports Errors

Vertica reports warnings and errors via two different mechanisms: SQLSTATEs and error messages. SQLSTATEs are intended for use by client applications, such as those accessing Vertica via ODBC or JDBC. Error messages are displayed to interactive users (for example, users connected to Vertica through vsql) and written to error logs.

About SQLSTATE

Vertica reports the success or failure of each statement it executes to client applications using a five-character SQLSTATE value. Many of these values are defined by the SQL standard. Others (identified by the letter "V" in their values) are Vertica-specific.

SQLSTATE values are grouped into classes which are defined by the first two characters in the SQLSTATE value. The last three characters indicate a specific condition within a class. For example, the SQLSTATE class 22 represents all data errors. The specific SQLSTATE value 22012 represents a division by zero error. SQLSTATE classes let an application that does not recognize a specific SQLSTATE value to still get a general idea of the result.

Warning and Error Messages

Each error and warning message displayed to interactive users or written to a log file by Vertica has its own numeric error code assigned to it. For example:

```
ERROR 3117: Division by zero  
WARNING 4098: No projections found  
ERROR 5617: Multiple WITH clauses not allowed
```

The error code number is not related to the SQLSTATE value. However, error and warning messages do correspond to a specific SQLSTATE. They are just a more-specific human-readable message compared to the SQLSTATE, which is mainly intended for client applications.

For example, all warning messages displayed by Vertica correspond to the SQLSTATE class 01. The warning message "WARNING 3084: Design Workspace couldn't be dropped" corresponds to the SQLSTATE value 01000 ERRCODE_WARNING.

Error codes do not change between Vertica releases, but individual error and warning messages may be added or removed in new releases. Your client application should not depend on particular error code appearing from one release to the next. Instead, it should use the SQLSTATE value to determine the result of executing a statement.

See the [SQL State List](#) for a list of all the SQLSTATE classes and values defined by Vertica. This table also links to lists of error or warning messages that are associated with each SQLSTATE value.

SQL State List

The following table lists the SQLSTATE classes and individual SQLSTATE codes.

SQLState	Description	Details
Class 00—Successful Completion		
00000	ERRCODE_SUCCESSFUL_COMPLETION	
Class 01—Warning		
01000	ERRCODE_WARNING	associated warning messages
01003	ERRCODE_WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION	
01004	ERRCODE_WARNING_STRING_DATA_RIGHT_TRUNCATION	associated warning messages
01006	ERRCODE_WARNING_PRIVILEGE_NOT_REVOKED	associated warning messages
01007	ERRCODE_WARNING_PRIVILEGE_NOT_GRANTED	associated warning messages
01008	ERRCODE_WARNING_PRIVILEGE_ALREADY_GRANTED	
01009	ERRCODE_WARNING_PRIVILEGE_ALREADY_REVOKED	
0100C	ERRCODE_WARNING_DYNAMIC_RESULT_SETS_RETURNED	
01V01	ERRCODE_WARNING_DEPRECATED_FEATURE	associated warning messages
01V02	ERRCODE_WARNING_QUERY_RETRIED	
Class 02—No Data		
02000	ERRCODE_NO_DATA	
02001	ERRCODE_NO_ADDITIONAL_DYNAMIC_RESULT_SETS_RETURNED	

SQLState	Description	Details
Class 03—SQL Statement Not Yet Complete		
03000	ERRCODE_SQL_STATEMENT_NOT_YET_COMPLETE	
Class 08—Client Connection Exception		
08000	ERRCODE_CONNECTION_EXCEPTION	associated error messages
08001	ERRCODE_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION	associated error messages
08003	ERRCODE_CONNECTION_DOES_NOT_EXIST	associated error messages
08004	ERRCODE_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION	
08006	ERRCODE_CONNECTION_FAILURE	associated error messages
08007	ERRCODE_TRANSACTION_RESOLUTION_UNKNOWN	
08V01	ERRCODE_PROTOCOL_VIOLATION	associated error messages
Class 09—Triggered Action Exception		
09000	ERRCODE_TRIGGERED_ACTION_EXCEPTION	associated error messages
Class 0A—Feature Not Supported		
0A000	ERRCODE_FEATURE_NOT_SUPPORTED	associated error messages
0A005	ERRCODE_PLAN_TO_SQL_NOT_SUPPORTED	associated error messages
Class 0B—Invalid Transaction Initiation		
0B000	ERRCODE_INVALID_TRANSACTION_INITIATION	associated error messages

SQLState	Description	Details
Class 0F—Locator Exception		
0F000	ERRCODE_LOCATOR_EXCEPTION	
0F001	ERRCODE_L_E_INVALID_SPECIFICATION	
Class 0L—Invalid Grantor		
0L000	ERRCODE_INVALID_GRANTOR	
0LV01	ERRCODE_INVALID_GRANT_OPERATION	associated error messages
Class 0P—Invalid Role Specification		
0P000	ERRCODE_INVALID_ROLE_SPECIFICATION	
Class 21—Cardinality Violation		
21000	ERRCODE_CARDINALITY_VIOLATION	
Class 22—Data Exception		
22000	ERRCODE_DATA_EXCEPTION	associated error messages
22001	ERRCODE_STRING_DATA_RIGHT_TRUNCATION	associated error messages
22002	ERRCODE_NULL_VALUE_NO_INDICATOR_PARAMETER	
22003	ERRCODE_NUMERIC_VALUE_OUT_OF_RANGE	associated error messages
22004	ERRCODE_NULL_VALUE_NOT_ALLOWED	associated error messages
22005	ERRCODE_ERROR_IN_ASSIGNMENT	
22007	ERRCODE_INVALID_DATETIME_FORMAT	associated error messages
22008	ERRCODE_DATETIME_FIELD_OVERFLOW	associated error messages

SQLState	Description	Details
22009	ERRCODE_INVALID_TIME_ZONE_DISPLACEMENT_VALUE	associated error messages
2200B	ERRCODE_ESCAPE_CHARACTER_CONFLICT	associated error messages
2200C	ERRCODE_INVALID_USE_OF_ESCAPE_CHARACTER	
2200D	ERRCODE_INVALID_ESCAPE_OCTET	associated error messages
2200F	ERRCODE_ZERO_LENGTH_CHARACTER_STRING	
2200G	ERRCODE_MOST_SPECIFIC_TYPE_MISMATCH	
22010	ERRCODE_INVALID_INDICATOR_PARAMETER_VALUE	
22011	ERRCODE_SUBSTRING_ERROR	associated error messages
22012	ERRCODE_DIVISION_BY_ZERO	associated error messages
22015	ERRCODE_INTERVAL_FIELD_OVERFLOW	associated error messages
22018	ERRCODE_INVALID_CHARACTER_VALUE_FOR_CAST	
22019	ERRCODE_INVALID_ESCAPE_CHARACTER	associated error messages
2201B	ERRCODE_INVALID_REGULAR_EXPRESSION	associated error messages
2201E	ERRCODE_INVALID_ARGUMENT_FOR_LOG	
2201F	ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION	
2201G	ERRCODE_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION	associated error messages
22020	ERRCODE_INVALID_LIMIT_VALUE	

SQLState	Description	Details
22021	ERRCODE_CHARACTER_NOT_IN_REPERTOIRE	associated error messages
22022	ERRCODE_INDICATOR_OVERFLOW	
22023	ERRCODE_INVALID_PARAMETER_VALUE	associated error messages
22024	ERRCODE_UNTERMINATED_C_STRING	
22025	ERRCODE_INVALID_ESCAPE_SEQUENCE	associated error messages
22026	ERRCODE_STRING_DATA_LENGTH_MISMATCH	
22027	ERRCODE_TRIM_ERROR	
2202E	ERRCODE_ARRAY_ELEMENT_ERROR	
22906	ERRCODE_NONSTANDARD_USE_OF_ESCAPE_CHARACTER	associated error messages
22V01	ERRCODE_FLOATING_POINT_EXCEPTION	
22V02	ERRCODE_INVALID_TEXT_REPRESENTATION	associated error messages
22V03	ERRCODE_INVALID_BINARY_REPRESENTATION	associated error messages
22V04	ERRCODE_BAD_COPY_FILE_FORMAT	associated error messages
22V05	ERRCODE_UNTRANSLATABLE_CHARACTER	
22V0B	ERRCODE_ESCAPE_CHARACTER_ON_NOESCAPE	associated error messages
22V21	ERRCODE_INVALID_EPOCH	associated error messages
22V22	ERRCODE_PLPGSQL_ERROR	
22V23	ERRCODE_RAISE_EXCEPTION	associated error

SQLState	Description	Details
		messages
22V24	ERRCODE_COPY_PARSE_ERROR	associated error messages
Class 23—Integrity Constraint Violation		
23000	ERRCODE_INTEGRITY_CONSTRAINT_VIOLATION	
23001	ERRCODE_RESTRICT_VIOLATION	
23502	ERRCODE_NOT_NULL_VIOLATION	associated error messages
23503	ERRCODE_FOREIGN_KEY_VIOLATION	associated error messages
23505	ERRCODE_UNIQUE_VIOLATION	associated error messages
23514	ERRCODE_CHECK_VIOLATION	associated error messages
Class 24—Invalid Cursor State		
24000	ERRCODE_INVALID_CURSOR_STATE	
Class 25—Invalid Transaction State		
25000	ERRCODE_INVALID_TRANSACTION_STATE	
25001	ERRCODE_ACTIVE_SQL_TRANSACTION	
25002	ERRCODE_BRANCH_TRANSACTION_ALREADY_ACTIVE	
25003	ERRCODE_INAPPROPRIATE_ACCESS_MODE_FOR_BRANCH_TRANSACTION	
25004	ERRCODE_INAPPROPRIATE_ISOLATION_LEVEL_FOR_BRANCH_TRANSACTION	
25005	ERRCODE_NO_ACTIVE_SQL_TRANSACTION_FOR_BRANCH_TRANSACTION	

SQLState	Description	Details
25006	ERRCODE_READ_ONLY_SQL_TRANSACTION	
25007	ERRCODE_SCHEMA_AND_DATA_STATEMENT_MIXING_NOT_SUPPORTED	
25008	ERRCODE_HELD_CURSOR_REQUIRES_SAME_ISOLATION_LEVEL	
25V01	ERRCODE_NO_ACTIVE_SQL_TRANSACTION	associated error messages
25V02	ERRCODE_IN_FAILED_SQL_TRANSACTION	
Class 26—Invalid SQL Statement Name		
26000	ERRCODE_INVALID_SQL_STATEMENT_NAME	
Class 27—Triggered Data Change Violation		
27000	ERRCODE_TRIGGERED_DATA_CHANGE_VIOLATION	
Class 28—Invalid Authorization Specification		
28000	ERRCODE_INVALID_AUTHORIZATION_SPECIFICATION	associated error messages
28001	ERRCODE_ACCOUNT_LOCKED	
28002	ERRCODE_PASSWORD_EXPIRED	
28003	ERRCODE_PASSWORD_IN_GRACE_PERIOD	
Class 2B—Dependent Privilege Descriptors Still Exist		
2B000	ERRCODE_DEPENDENT_PRIVILEGE_DESCRIPTORSTILL_EXIST	
2BV01	ERRCODE_DEPENDENT_OBJECTS_STILL_EXIST	associated error messages
Class 2D—Invalid Transaction Termination		
2D000	ERRCODE_INVALID_TRANSACTION_TERMINATION	
Class 2F—SQL Routine Exception		

SQLState	Description	Details
2F000	ERRCODE_SQL_ROUTINE_EXCEPTION	
2F002	ERRCODE_S_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED	
2F003	ERRCODE_S_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED	
2F004	ERRCODE_S_R_E_READING_SQL_DATA_NOT_PERMITTED	
2F005	ERRCODE_S_R_E_FUNCTION_EXECUTED_NO_RETURN_STATEMENT	
Class 34—Invalid Cursor Name		
34000	ERRCODE_INVALID_CURSOR_NAME	
Class 38—External Routine Exception		
38000	ERRCODE_EXTERNAL_ROUTINE_EXCEPTION	
38001	ERRCODE_E_R_E_CONTAINING_SQL_NOT_PERMITTED	
38002	ERRCODE_E_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED	
38003	ERRCODE_E_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED	
38004	ERRCODE_E_R_E_READING_SQL_DATA_NOT_PERMITTED	associated error messages
Class 39—External Routine Invocation Exception		
39000	ERRCODE_EXTERNAL_ROUTINE_INVOCATION_EXCEPTION	
39001	ERRCODE_E_R_I_E_INVALID_SQLSTATE_RETURNED	
39004	ERRCODE_E_R_I_E_NULL_VALUE_NOT_ALLOWED	
39V01	ERRCODE_E_R_I_E_TRIGGER_PROTOCOL_VIOLATED	

SQLState	Description	Details
39V02	ERRCODE_E_R_I_E_SRF_PROTOCOL_VIOLATED	
Class 3B—Savepoint Exception		
3B000	ERRCODE_SAVEPOINT_EXCEPTION	
3B001	ERRCODE_S_E_INVALID_SPECIFICATION	
Class 3D—Invalid Catalog Name		
3D000	ERRCODE_INVALID_CATALOG_NAME	
Class 3F—Invalid Schema Name		
3F000	ERRCODE_INVALID_SCHEMA_NAME	
Class 40—Transaction Rollback		
40000	ERRCODE_TRANSACTION_ROLLBACK	
40001	ERRCODE_T_R_SERIALIZATION_FAILURE	
40002	ERRCODE_T_R_INTEGRITY_CONSTRAINT_VIOLATION	
40003	ERRCODE_T_R_STATEMENT_COMPLETION_UNKNOWN	
40V01	ERRCODE_T_R_DEADLOCK_DETECTED	associated error messages
Class 42—Syntax Error or Access Rule Violation		
42000	ERRCODE_SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION	
42501	ERRCODE_INSUFFICIENT_PRIVILEGE	associated error messages
42601	ERRCODE_SYNTAX_ERROR	associated error messages
42602	ERRCODE_INVALID_NAME	associated error messages
42611	ERRCODE_INVALID_COLUMN_DEFINITION	associated error messages

SQLState	Description	Details
42622	ERRCODE_NAME_TOO_LONG	associated error messages
42701	ERRCODE_DUPLICATE_COLUMN	associated error messages
42702	ERRCODE_AMBIGUOUS_COLUMN	associated error messages
42703	ERRCODE_UNDEFINED_COLUMN	associated error messages
42704	ERRCODE_UNDEFINED_OBJECT	associated error messages
42710	ERRCODE_DUPLICATE_OBJECT	associated error messages
42712	ERRCODE_DUPLICATE_ALIAS	associated error messages
42723	ERRCODE_DUPLICATE_FUNCTION	associated error messages
42725	ERRCODE_AMBIGUOUS_FUNCTION	associated error messages
42803	ERRCODE_GROUPING_ERROR	associated error messages
42804	ERRCODE_DATATYPE_MISMATCH	associated error messages
42809	ERRCODE_WRONG_OBJECT_TYPE	associated error messages
42830	ERRCODE_INVALID_FOREIGN_KEY	associated error messages
42846	ERRCODE_CANNOT_COERCE	associated error messages
42883	ERRCODE_UNDEFINED_FUNCTION	associated error

SQLState	Description	Details
		messages
42939	ERRCODE_RESERVED_NAME	associated error messages
42P20	ERRCODE_WINDOWING_ERROR	associated error messages
42V01	ERRCODE_UNDEFINED_TABLE	associated error messages
42V02	ERRCODE_UNDEFINED_PARAMETER	associated error messages
42V03	ERRCODE_DUPLICATE_CURSOR	associated error messages
42V04	ERRCODE_DUPLICATE_DATABASE	associated error messages
42V05	ERRCODE_DUPLICATE_PSTATEMENT	
42V06	ERRCODE_DUPLICATE_SCHEMA	associated error messages
42V07	ERRCODE_DUPLICATE_TABLE	associated error messages
42V08	ERRCODE_AMBIGUOUS_PARAMETER	associated error messages
42V09	ERRCODE_AMBIGUOUS_ALIAS	associated error messages
42V10	ERRCODE_INVALID_COLUMN_REFERENCE	associated error messages
42V11	ERRCODE_INVALID_CURSOR_DEFINITION	associated error messages
42V12	ERRCODE_INVALID_DATABASE_DEFINITION	
42V13	ERRCODE_INVALID_FUNCTION_DEFINITION	associated error messages

SQLState	Description	Details
42V14	ERRCODE_INVALID_PSTATEMENT_DEFINITION	
42V15	ERRCODE_INVALID_SCHEMA_DEFINITION	associated error messages
42V16	ERRCODE_INVALID_TABLE_DEFINITION	associated error messages
42V17	ERRCODE_INVALID_OBJECT_DEFINITION	associated error messages
42V18	ERRCODE_INDETERMINATE_DATATYPE	associated error messages
42V21	ERRCODE_UNDEFINED_PROJECTION	associated error messages
42V22	ERRCODE_UNDEFINED_NODE	
42V23	ERRCODE_UNDEFINED_PERMUTATION	
42V24	ERRCODE_UNDEFINED_USER	associated error messages
42V25	ERRCODE_PATTERN_MATCH_ERROR	associated error messages
42V26	ERRCODE_DUPLICATE_NODE	associated error messages
Class 44—WITH CHECK OPTION Violation		
44000	ERRCODE_WITH_CHECK_OPTION_VIOLATION	
Class 53—Insufficient Resources		
53000	ERRCODE_INSUFFICIENT_RESOURCES	associated error messages
53100	ERRCODE_DISK_FULL	associated error messages
53200	ERRCODE_OUT_OF_MEMORY	associated error messages

SQLState	Description	Details
53300	ERRCODE_TOO_MANY_CONNECTIONS	
Class 54—Program Limit Exceeded		
54000	ERRCODE_PROGRAM_LIMIT_EXCEEDED	associated error messages
54001	ERRCODE_STATEMENT_TOO_COMPLEX	associated error messages
54011	ERRCODE_TOO_MANY_COLUMNS	associated error messages
54023	ERRCODE_TOO_MANY_ARGUMENTS	associated error messages
Class 55—Object Not In Prerequisite State		
55000	ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE	associated error messages
55006	ERRCODE_OBJECT_IN_USE	associated error messages
55V02	ERRCODE_CANT_CHANGE_RUNTIME_PARAM	associated error messages
55V03	ERRCODE_LOCK_NOT_AVAILABLE	associated error messages
55V04	ERRCODE_TM_MARKER_NOT_AVAILABLE	associated error messages
Class 57—Operator Intervention		
57000	ERRCODE_OPERATOR_INTERVENTION	
57014	ERRCODE_QUERY_CANCELED	associated error messages
57015	ERRCODE_SLOW_DELETE	associated error messages
57V01	ERRCODE_ADMIN_SHUTDOWN	associated error

SQLState	Description	Details
		messages
57V02	ERRCODE_CRASH_SHUTDOWN	
57V03	ERRCODE_CANNOT_CONNECT_NOW	associated error messages
57V04	ERRCODE_DML_COMMIT_DURING_SHUTDOWN	associated error messages
Class 58—System Error		
58030	ERRCODE_IO_ERROR	associated error messages
58V01	ERRCODE_UNDEFINED_FILE	associated error messages
58V02	ERRCODE_DUPLICATE_FILE	
Class V1—Vertica-specific multi-node errors class		
V1001	ERRCODE_LOST_CONNECTIVITY	associated error messages
V1002	ERRCODE_K_SAFETY_VIOLATION	associated error messages
V1003	ERRCODE_CLUSTER_CHANGE	associated error messages
Class V2—Vertica-specific miscellaneous errors class		
V2000	ERRCODE_AUTH_FAILED	associated error messages
V2001	ERRCODE_LICENSE_ISSUE	associated error messages
V2002	ERRCODE_MOVEOUT_ABORTED	
Class VC—Configuration File Error		
VC001	ERRCODE_CONFIG_FILE_ERROR	associated error

SQLState	Description	Details
		messages
VC002	ERRCODE_LOCK_FILE_EXISTS	
Class VD—DB Designer errors		
VD001	ERRCODE_DESIGNER_FUNCTION_ERROR	associated error messages
Class VP—User procedure errors		
VP000	ERRCODE_USER_PROC_ERROR	associated error messages
VP001	ERRCODE_USER_PROC_EXEC_ERROR	associated error messages
Class VX—Internal Error		
VX001	ERRCODE_INTERNAL_ERROR	associated error messages
VX002	ERRCODE_DATA_CORRUPTED	associated error messages
VX003	ERRCODE_INDEX_CORRUPTED	associated error messages
VX004	ERRCODE_PLAN_TO_SQL_INTERNAL_EROR	associated error messages

Warning Messages Associated with SQLSTATE 01000

This topic lists the warnings associated with the SQLSTATE 01000.

SQLSTATE 01000 Description

ERRCODE_WARNING

Warning messages associated with this SQLState

WARNING 2021: *string* Directory for errors files was not created.
Unable to write errors for this instance of COPY command

WARNING 2022: *string* Directory for exceptions files was not created.
Unable to write errors for this instance of COPY command

WARNING 2023: *string* Directory for rejected data files was not created.
Unable to write errors for this instance of COPY command

WARNING 2362: Cannot begin transaction; transaction is already running

WARNING 3084: Design couldn't be dropped

WARNING 3152: Duplicate values in columns marked as UNIQUE will now be ignored for the remainder of your session or until `reenable_duplicate_key_error()` is called

WARNING 3372: Failed to disable profiling: *string*

WARNING 3373: Failed to enable profiling: *string*

WARNING 3539: Incorrect results are possible. Please contact Vertica Support if unsure

WARNING 3791: Invalid view *string*: *string*

WARNING 4071: NO COMMIT option will be ignored for external table "*string*"

WARNING 4088: No new valid default roles specified. Retaining previous set of default roles for user *string*

WARNING 4098: No projections found

WARNING 4102: No rows are inserted into table "*string*". "*string*" because ON COMMIT DELETE ROWS is the default for create temporary table

WARNING 4116: No super projections created for table *string*.

WARNING 4246: Only GLOBAL scope is supported for clearing *string* profiles

WARNING 4463: Projection *string* is not up to date

WARNING 4468: Projection `<string>` is not available for query processing. Execute the `select start_refresh()` function to copy data into this projection.
The projection must have a sufficient number of buddy projections and all nodes must be up before starting a refresh

WARNING 4792: Storage option "*string*" will be ignored for external table "*string*"

WARNING 4871: System view *string* for tuning rule *string* is currently invalid

WARNING 4873: System view for tuning rule *string* does not exist

WARNING 4996: This request may deadlock the system. Please report the details to technical support

WARNING 5068: Total declared length of columns of one of the constraints exceeds the limit, truncation may happen

WARNING 5119: UDX code didn't respond when Vertica tried to get function prototype for *string* in library *string*: *string*

WARNING 5448: View *string* is currently invalid

WARNING 5451: Violations of some of foreign key constraints may not be reported because of no privilege on the foreign tables

WARNING 5642: Projection *string* is not persistent or not up to date; it will not be copied

WARNING 5643: Projection *string* is prejoin projection; it will not be copied

WARNING 5717: No statistics has been exported. Either the DB is empty or you try to export an external table or you do not have access to the available objects

Vertica Documentation

Vertica Error Messages

WARNING 5724: Segmentation clause contains a *string* - data loads may be slowed significantly

WARNING 5727: Sort clause contains a *string* - data loads may be slowed significantly

WARNING 5741: View *string* depends on other relations

WARNING 5819: Design could not be reset

WARNING 5821: Detected keys sharing the same case-insensitive key name

WARNING 5860: Due to the data isolation of temp tables with an on-commit-delete-rows policy, the `compute_flextable_keys()` and `compute_flextable_keys_and_build_view()` functions cannot access this table's data. The `build_flextable_view()` function can be used with a user-provided keys table to create a view, but involves a DDL commit which will delete the table's rows

WARNING 5873: Failed to add table *string* of hcatalog schema *string* to catalog: *string*

WARNING 5875: Failed to alter table *string* of hcatalog schema *string* to catalog: *string*

WARNING 5880: Failed to describe table *string* in hcatalog database *string*: *string*

WARNING 5881: Failed to describe table *string* in schema *string*: HCatalog database *string* does not exist

WARNING 5884: Failed to list hcatalog tables of hcatalog schema *string*: *string*

WARNING 5886: Failed to mirror table *string* in schema *string*: *string*

WARNING 5909: Found and ignored keys with names longer than the maximum column-name length limit

WARNING 5912: HASH() arguments contain irregular expressions

WARNING 5917: Ignored some keys since the total key count exceeds the view column limit

WARNING 5922: Insufficient privileges to alter table *string*

WARNING 5923: Insufficient privileges to drop table *string*

WARNING 5991: Projection basename "*string*" hint was ignored. "*string*" is used as the basename

WARNING 5993: Projection is irregularly segmented by column

WARNING 6053: The view creation involved a DDL commit which deleted the table's rows

WARNING 6256: Error while analyzing constraint(s) *string* on *string*

WARNING 6257: Error while creating LTT for analyzing constraints on *string*

WARNING 6356: No partitions have been swapped. Neither table has partitions fall in range

WARNING 6417: Swapped partitions are not immediately moved to a new storage location

WARNING 6515: Projection with name "*string*" already exists in schema "*string*" for anchor table "*string.string*"

WARNING 6565: Inequal query trees produced the same hash code query1:
string
query2:
string

WARNING 6594: Unknown RangeTblEntry: *value*

WARNING 6608: *string* is an invalid argument for hint *string*

WARNING 6765: Error parsing distrib value *string* in Distrib hint's arguments. The whole hint will be ignored

WARNING 6773: Failed to compare plans for query '*string*'

WARNING 6797: Hint *string* can accept at most one argument, the hint with *value* arguments is ignored

WARNING 6798: Hint *string* can be specified only once for each join, if multiple instances listed only the first is considered

WARNING 6799: Hint *string* is not feasible and will be ignored

WARNING 6801: Hint *string* requires exactly two arguments, the hint with *value* arguments is ignored

WARNING 6813: Inherited privileges are globally disabled; schema parameter is set but has no effect

WARNING 6814: Inherited privileges are globally disabled; table parameter is set but has no effect

Vertica Documentation

Vertica Error Messages

WARNING 6815: Inherited privileges are globally disabled; view parameter is set but has no effect

WARNING 6818: Input operations specified for Hint *string* is not feasible and will be ignored

WARNING 6922: Projection name was changed to *string* because it conflicts with the basename of the table *string*

WARNING 6990: Text index has invalid tokenizer

WARNING 6991: The active saved plan for this query has incompatible output column set. Continuing with the original query

WARNING 7000: The projection '*string*' was used to enforce the enabled key constraint '*string*', and may be regenerated to validate a DML statement on the base table

WARNING 7020: Unable to find the following query in the export file: '*string*'

WARNING 7030: Unexpected group clause found during query comparison

WARNING 7031: Unexpected group clause found during query hashing

WARNING 7033: Unexpected sort clause found during query comparison

WARNING 7036: Unknown Expr:*value* during query comparison

WARNING 7037: Unknown Expr:*value* found

WARNING 7038: Unknown Expr:*value* found during query hashing

WARNING 7039: Unknown Inequality OpOid:*value* found during query hashing

WARNING 7040: Unknown Logic OpExpr:*value* found during query comparison

WARNING 7041: Unknown Logic OpExpr:*value* found during query hashing

WARNING 7044: Unknown OpExpr:*value* found during query comparison

WARNING 7045: Unknown OpExpr:*value* found during query hashing

WARNING 7046: Unknown RangeTblEntry:*value* found during query comparison

WARNING 7047: Unknown RangeTblEntry:*value* found during query hashing

WARNING 7048: Unknown Sublink:*value* found during query comparison

WARNING 7049: Unknown Sublink:*value* found during query hashing

WARNING 7052: Unsupported JoinExpr Type found during query comparison

WARNING 7053: Unsupported JoinExpr Type found during query hashing

WARNING 7070: View "*string*" will include privileges from schema "*string*"

WARNING 7074: You appear to be using an old version of HDFS that may have stability problems under high load. See the Vertica documentation for supported HDFS versions

WARNING 7186: Inherited privileges are globally disabled. Privileges will be materialized and visibility of table *string* may change

WARNING 7204: Updates on the target table with a prejoin projection might ignore PK constraints and cause data integrity violations

WARNING 7265: Inherited privileges are globally disabled. Privileges will be materialized and visibility of view *string* may change

WARNING 7709: Could not drop table '*string*'. Please remove manually if necessary.
Detail: *string*

WARNING 7752: Cannot make a local query plan for "try local" hint, please check hash function, segment quantity, segment boundary, etc

WARNING 7793: The cache will be set to 1 instead

WARNING 7942: Unknown expression found during hashing: *value*

WARNING 7994: Not converged before max_iterations [*value*]

WARNING 8012: The parameter [*string*] is not defined for this function

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Warning Messages Associated with SQLSTATE 01004

This topic lists the warnings associated with the SQLSTATE 01004.

SQLSTATE 01004 Description

ERRCODE_WARNING_STRING_DATA_RIGHT_TRUNCATION

Warning messages associated with this SQLState

WARNING 7166: client_label exceeded maximum length; truncated to 255 characters

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Warning Messages Associated with SQLSTATE 01006

This topic lists the warnings associated with the SQLSTATE 01006.

SQLSTATE 01006 Description

ERRCODE_WARNING_PRIVILEGE_NOT_REVOKED

Warning messages associated with this SQLState

WARNING 4925: The string "string" cannot be string string "string"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Warning Messages Associated with SQLSTATE 01007

This topic lists the warnings associated with the SQLSTATE 01007.

SQLSTATE 01007 Description

ERRCODE_WARNING_PRIVILEGE_NOT_GRANTED

Warning messages associated with this SQLState

WARNING 5682: USAGE privilege on schema "*string*" also needs to be granted to "*string*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Warning Messages Associated with SQLSTATE 01V01

This topic lists the warnings associated with the SQLSTATE 01V01.

SQLSTATE 01V01 Description

ERRCODE_WARNING_DEPRECATED_FEATURE

Warning messages associated with this SQLState

WARNING 2693: Configuration parameter *string* has been deprecated; setting it has no effect
WARNING 4736: *set_local_segment_threshold* has been deprecated; setting it has no effect

WARNING 7399: Created prejoin projection '*string*'. Prejoin projections have been deprecated and will be removed in version 9.0

WARNING 7585: Using prejoin projection '*string*'. Prejoin projections have been deprecated and will be removed in version 9.0

WARNING 7768: HDFS Connector is deprecated. Use 'COPY FROM' syntax directly with 'hdfs://' scheme URLs

WARNING 7905: This API for optional parameters is deprecated

WARNING 8015: This API for deleting model is deprecated

WARNING 8016: This API for renaming model is deprecated

WARNING 8030: The owner parameter in `summarize_model` is deprecated

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 08000

This topic lists the errors associated with the SQLSTATE 08000.

SQLSTATE 08000 Description

ERRCODE_CONNECTION_EXCEPTION

Error messages associated with this SQLState

ERROR 2029: *string* from stdin failed: *string*

ERROR 2708: Connection to database [*string*] is invalid

ERROR 2896: Could not receive data from server:*string*

ERROR 2908: Could not send data to server: *string*

ERROR 3276: Error while waiting on socket. *value*

ERROR 4342: Password encryption failed

ERROR 5197: Unknown authentication method (*value*) requested by server

ERROR 7662: MD5 password hash requested when MD5 is not allowed

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 08001

This topic lists the errors associated with the SQLSTATE 08001.

SQLSTATE 08001 Description

ERRCODE_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION

Error messages associated with this SQLState

ERROR 2322: Cancel() -- connect() failed:
ERROR 2324: Cancel() -- socket() failed:
ERROR 2823: Could not connect to server [*string*]: *string*
Is the server running and accepting
TCP/IP connections on port *string*?
ERROR 2824: Could not connect to server: *string*
Is the server running on host [*string*] and accepting
TCP/IP connections on port *string*?
ERROR 2839: Could not create socket: *string*
ERROR 2865: Could not get client address from socket: *string*
ERROR 2869: Could not get socket error status: *string*
ERROR 2912: Could not set socket to close-on-exec mode: *string*
ERROR 2913: Could not set socket to non-blocking mode: *string*
ERROR 2914: Could not set socket to TCP no delay mode: *string*
ERROR 7801: Could not translate host name "*string*" to address using family "*string*": *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 08003

This topic lists the errors associated with the SQLSTATE 08003.

SQLSTATE 08003 Description

ERRCODE_CONNECTION_DOES_NOT_EXIST

Error messages associated with this SQLState

ERROR 4717: Server closed the connection unexpectedly
This probably means the server terminated abnormally
before or while processing the request

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 08006

This topic lists the errors associated with the SQLSTATE 08006.

SQLSTATE 08006 Description

ERRCODE_CONNECTION_FAILURE

Error messages associated with this SQLState

ERROR 2323: Cancel() -- send() failed: *string*
ERROR 2606: Client failed when looking for pending signals
ERROR 4539: Received no response from *stringstring*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 08V01

This topic lists the errors associated with the SQLSTATE 08V01.

SQLSTATE 08V01 Description

ERRCODE_PROTOCOL_VIOLATION

Error messages associated with this SQLState

ERROR 2055: *string* Unexpected message type *string* reading from stdin

ERROR 2257: Bind message has *value* parameter formats but *value* parameters

ERROR 2258: Bind message has *value* result formats but query has *value* columns

ERROR 3334: Expected a RowDescription Message

ERROR 3335: Expected a SendExport Message

ERROR 3575: Insufficient data left in message

ERROR 3631: Invalid CLOSE message subtype *value*

ERROR 3651: Invalid DESCRIBE message subtype *value*

ERROR 3699: Invalid message format

ERROR 3701: Invalid message type

ERROR 3702: Invalid message type *value*

ERROR 3755: Invalid string in message

ERROR 3887: Lost synchronization with server: length *value*

ERROR 4074: No data left in message

ERROR 4718: Server did not identify with a pid & key

ERROR 5181: Unexpected message type *0xhex value*

ERROR 5208: Unknown message from server

ERROR 5872: Expected to flush an end-of-batch client message but received a message of type *value*.
Attempting to recover...

ERROR 6863: MARS operation not supported for your client version. Parameter not changed

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 09000

This topic lists the errors associated with the SQLSTATE 09000.

SQLSTATE 09000 Description

ERRCODE_TRIGGERED_ACTION_EXCEPTION

Error messages associated with this SQLState

- ERROR 7353: A problem during the execution of writeModel.
Detail: *string*
- ERROR 7354: A problem in the kmeans summary.
Detail: *string*
- ERROR 7355: A problem in the regression summary.
Detail: *string*
- ERROR 7356: A problem occurred during the execution of a BFGS iteration.
Detail: *string*
- ERROR 7359: A problem occurred during the execution of a Newton iteration.
Detail: *string*
- ERROR 7362: A problem occurred during the execution of an iteration.
Detail: *string*
- ERROR 7373: Cannot check if all columns are numeric.
Detail: *string*
- ERROR 7374: Cannot check if it is a table.
Detail: *string*
- ERROR 7375: Cannot check if it is a view.
Detail: *string*
- ERROR 7377: Cannot compute input column list.
Detail: *string*
- ERROR 7378: Cannot find the current user.
Detail: *string*
- ERROR 7380: Cannot get column names.
Detail: *string*
- ERROR 7381: Cannot get column types.
Detail: *string*
- ERROR 7383: Cannot make inf clause.
Detail: *string*
- ERROR 7384: Cannot make null clause.
Detail: *string*

Vertica Documentation

Vertica Error Messages

ERROR 7385: Cannot parse tablename.
Detail: *string*

ERROR 7395: Could not compute evaluation metrics.
Detail: *string*

ERROR 7396: Could not create view '*string*'.
Detail: *string*

ERROR 7497: Problem in balance.
Detail: *string*

ERROR 7498: Problem in balance: Unknown exception

ERROR 7499: Problem in delete_model.
Detail: *string*

ERROR 7502: Problem in deleting the model.
Detail: *string*

ERROR 7503: Problem in detect_outliers.
Detail: *string*

ERROR 7504: Problem in detect_outliers: Unknown exception

ERROR 7505: Problem in initializing centers table.
Detail: *string*

ERROR 7506: Problem in kmeans.
Detail: *string*

ERROR 7507: Problem in kmeans: Unknown exception

ERROR 7508: Problem in linear_reg.
Detail: *string*

ERROR 7509: Problem in linear_reg: Unknown exception

ERROR 7512: Problem in logistic_reg.
Detail: *string*

ERROR 7513: Problem in logistic_reg: Unknown exception

ERROR 7516: Problem in normalize.
Detail: *string*

ERROR 7517: Problem in normalize: Unknown exception

ERROR 7518: Problem in parsing the optional arguments.
Detail: *string*

ERROR 7519: Problem in reading the information of the model.
Detail: *string*

ERROR 7520: Problem in rename_model.
Detail: *string*

ERROR 7522: Problem in renaming the model.
Detail: *string*

ERROR 7523: Problem in summarize_model.
Detail: *string*

ERROR 7524: Problem in writing initial model to DFS.
Detail: *string*

ERROR 7593: Could not randomly pick *value* distinct centers after trying *value* times

ERROR 7609: Problem in delete_model: Unknown Exception

ERROR 7610: Problem in rename_model: Unknown Exception

ERROR 7611: Problem in summarize_model: Unknown Exception

ERROR 7621: The old model name and new model name cannot be the same

Vertica Documentation

Vertica Error Messages

- ERROR 7626: A problem during the execution of computing the number of detected outliers.
Detail: *string*
- ERROR 7731: Problem in calculating alpha for Linear Regression.
Detail: *string*
- ERROR 7732: Problem in calculating Hessian matrix for Linear Regression.
Detail: *string*
- ERROR 7733: Problem in strongWolfeLineSearch.
Detail: *string*
- ERROR 7734: Problem in zoom.
Detail: *string*
- ERROR 7761: A problem during the execution of computing the mad.
Detail: *string*
- ERROR 7762: A problem during the execution of computing the median.
Detail: *string*
- ERROR 7763: A problem occurred during the execution of a mad computation.
Detail: *string*
- ERROR 7764: A problem occurred during the execution of median computation.
Detail: *string*
- ERROR 7769: Kmeans++ exceeded max number of retries for choosing initial center
- ERROR 7779: A problem during the execution of internal_predict_linearReg
- ERROR 7784: Could not remove blob named *string*.
Detail: Unexpected exception
- ERROR 7785: Could not remove the blob named *string*.
Detail: *string*
- ERROR 7796: A problem in the classifier summary.
Detail: *string*
- ERROR 7797: A problem occurred during training of Naive Bayes model.
Detail: *string*
- ERROR 7805: No input columns provided
- ERROR 7807: Problem in naive_bayes.
Detail: *string*
- ERROR 7808: Problem in naive_bayes: Unknown exception
- ERROR 7825: line_search_logistic failed
- ERROR 7826: Numeric overflow occurred during execution of kmeans++
- ERROR 7827: Numeric overflow occurred when computing total sum of squares
- ERROR 7828: Numeric overflow occurred when computing total within-cluster sum of squares
- ERROR 7829: Numeric overflow occurred when computing within-cluster sum of squares
- ERROR 7844: Input table *string* is empty
- ERROR 7845: No input columns left after excluding
- ERROR 7851: No rows remain after filtering rows with null and infinity values
- ERROR 7858: Only found *value* non-empty clusters. You may want to try again or use a better set of initial centers; or there may be fewer than k distinct datapoints in the table
- ERROR 7949: A problem occurred during the execution
of mode computation.
Detail: *string*
- ERROR 7961: Problem in impute.
Detail: *string*
- ERROR 7962: Problem in impute: Unknown exception

ERROR 7975: A problem occurred during training of SVM model.
Detail: *string*

ERROR 7988: Input table is empty

ERROR 7990: Invalid value [*string*] of parameter [*string*]

ERROR 7992: No rows remain after filtering rows containing NULL, NaN or INF

ERROR 8000: Problem in normalize_fit.
Detail: *string*

ERROR 8001: Problem in normalize_fit: Unknown exception

ERROR 8002: Problem in svm_classifier.
Detail: *string*

ERROR 8003: Problem in svm_classifier: Unknown exception

ERROR 8014: There was a problem reading the model summary.
Detail: *string*

ERROR 8020: Unexpected empty result from query

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 0A000

This topic lists the errors associated with the SQLSTATE 0A000.

SQLSTATE 0A000 Description

ERRCODE_FEATURE_NOT_SUPPORTED

Error messages associated with this SQLState

ERROR 2009: *string* can not be used in function *string*

ERROR 2013: *string* clause is not supported for expressions

ERROR 2014: *string* Concatenated GZIP/BZIP is not supported with NATIVE/NATIVE VARCHAR formats

ERROR 2036: *string* is not a legal time unit

ERROR 2058: *string* VIEW is not supported

ERROR 2089: A correlated column in a subquery expression is not supported

ERROR 2114: ADD COLUMN over temporary tables is not supported

ERROR 2130: Aggregate function *string* (*value*) is not supported

ERROR 2133: Aggregate function calls cannot contain subqueries

Vertica Documentation

Vertica Error Messages

ERROR 2138: Aggregate functions can only be called on columns of a table

ERROR 2161: ALL subquery with a correlated expression is not supported

ERROR 2165: ALTER COLUMN TYPE over temporary tables is not supported

ERROR 2166: ALTER TABLE does not support ADD COLUMN with other clauses

ERROR 2167: ALTER TABLE does not support ALTER COLUMN TYPE with other clauses

ERROR 2168: ALTER TABLE does not support DROP COLUMN with other clauses

ERROR 2169: ALTER TABLE does not support SET SCHEMA with other clauses

ERROR 2178: An expression containing a correlated subquery with aggregate function is not supported

ERROR 2183: Analytic functions are not allowed in an ORDER BY on a UNION/INTERSECT/EXCEPT

ERROR 2184: Analytic functions are not supported in the ORDER BY of an analytic function OVER clause

ERROR 2190: Analytics query with having clause expression that involves aggregates and subquery is not supported

ERROR 2192: ANALYZE_CONSTRAINTS is currently not supported in non-default locales

ERROR 2208: Another Design/Deployment is in progress

ERROR 2210: ANTI join with segmented inner not supported

ERROR 2220: Argument *string* must not contain subqueries

ERROR 2226: Argument to seeded random_must be a constant

ERROR 2233: Array References are not supported

ERROR 2235: ArrayExpr is not supported

ERROR 2329: Cannot accept a value of type any

ERROR 2330: Cannot accept a value of type anyarray

ERROR 2331: Cannot accept a value of type anyelement

ERROR 2332: Cannot accept a value of type internal

ERROR 2333: Cannot accept a value of type language_handler

ERROR 2334: Cannot accept a value of type opaque

ERROR 2335: Cannot accept a value of type trigger

ERROR 2340: Cannot add IDENTITY/AUTO-INCREMENT columns

ERROR 2350: Cannot alter type of column "*string*" since it is referenced in the constraint "*string*"

ERROR 2351: Cannot alter type of column "*string*" since it is referenced in the default expression of column "*string*"

ERROR 2352: Cannot alter type of column "*string*" since it is referenced in the partition expression

ERROR 2353: Cannot alter type of column "*string*" since it is referenced in the segmentation expression of projection "*string*"

ERROR 2354: Cannot alter type of column with a default expression

ERROR 2360: Cannot assign to system column "*string*"

ERROR 2363: Cannot broadcast non-subquery outer input to a join

ERROR 2368: Cannot change owner of temporary table

ERROR 2377: Cannot convert column "*string*" from "*string*" to type "*string*"

ERROR 2392: Cannot delete from a view

ERROR 2399: Cannot display a value of type any

ERROR 2400: Cannot display a value of type anyelement

ERROR 2401: Cannot display a value of type internal

ERROR 2402: Cannot display a value of type language_handler

Vertica Documentation

Vertica Error Messages

ERROR 2403: Cannot display a value of type opaque

ERROR 2404: Cannot display a value of type trigger

ERROR 2407: Cannot drop a table column when a node is down

ERROR 2411: Cannot drop column "string" since it is referenced in the partition expression

ERROR 2412: Cannot drop column "string" since it is referenced in the primary key constraint

ERROR 2425: Cannot export virtual *string string*

ERROR 2443: Cannot insert into a view

ERROR 2458: Cannot mergeout uncommitted data in the presence of savepoints

ERROR 2461: Cannot moveout uncommitted data in the presence of savepoints

ERROR 2503: Cannot set a subfield to DEFAULT

ERROR 2504: Cannot set an array element to DEFAULT

ERROR 2532: Cannot update a view

ERROR 2533: Cannot Update/Merge tables involved in prejoins with Limit clause on unsorted output

ERROR 2549: Cannot use DISTINCT with user-defined transform functions

ERROR 2552: Cannot use meta function or non-deterministic function in PARTITION BY expression

ERROR 2556: Cannot use SAVEPOINT with uncommitted tuple mover enabled

ERROR 2558: Cannot use subquery in EXECUTE parameter

ERROR 2559: Cannot use subquery in expressions within COPY

ERROR 2560: Cannot use subquery in PARTITION BY expression

ERROR 2561: Cannot use subquery in SEGMENTED BY expression

ERROR 2562: Cannot use Vertica's built-in file source and a UDSOURCE in the same query

ERROR 2569: Catalog object *string* does not exist

ERROR 2602: Clause "NO PROJECTION" conflicts with the column list

ERROR 2603: Clause "NO PROJECTION" is supported only on temporary tables

ERROR 2618: CoerceToDomain is not supported

ERROR 2619: CoerceToDomainValue is not supported

ERROR 2628: Column "string" in PARTITION BY expression is not allowed, since it contains NULL values

ERROR 2646: Column *string* has the NOT NULL constraint set and has no default value defined

ERROR 2648: Column *string* in PARTITION BY expression is not allowed, since it is not present in some projections

ERROR 2649: Column *string* in PARTITION BY expression is not allowed, since it may contain NULL values

ERROR 2652: Column *string* occurred multiple times in the definition of Projection *string*

ERROR 2660: Column *column string* is no longer at position *value* in table *string*

ERROR 2667: Column name list is not allowed in CREATE TABLE / AS EXECUTE

ERROR 2672: Column type int2 is not supported

ERROR 2673: Column type int4 is not supported

ERROR 2676: Command *string* is not supported

ERROR 2679: COMMENT not supported for system objects

ERROR 2680: COMMENT not supported for this object type

ERROR 2692: Conditional UNION/INTERSECT/EXCEPT statements are not implemented

ERROR 2698: Conflicting or redundant column options

ERROR 2721: ConvertRowtypeExpr is not supported

Vertica Documentation

Vertica Error Messages

ERROR 2725: Copy cannot return rejected rows from executor nodes

ERROR 2726: Copy cannot return rejected rows from more than one file

ERROR 2739: COPY force not null is available only in CSV mode, but CSV mode is not supported

ERROR 2740: COPY force quote is available only in CSV mode, but CSV mode is not supported

ERROR 2741: COPY FROM does not support the BINARY option

ERROR 2742: COPY FROM does not support the CSV option

ERROR 2743: COPY FROM does not support the OIDS option

ERROR 2744: COPY LOCAL does not support rejected row numbers with exceptions or rejected data options

ERROR 2751: COPY quote is available only in CSV mode, but CSV mode is not supported

ERROR 2770: Correlated EXISTS/NOT EXISTS subquery containing having clause with aggregates is not supported

ERROR 2772: Correlated EXISTS/NOT EXISTS subquery with limit 0 is not supported

ERROR 2773: Correlated EXISTS/NOT EXISTS with aggregate COUNT is not supported

ERROR 2776: Correlated EXISTS/NOT EXISTS with User Defined Aggregate is not supported

ERROR 2777: Correlated expression in ON clause is not supported

ERROR 2778: Correlated expression in set operator subquery is not supported

ERROR 2779: Correlated expressions in SELECT list of subquery are not supported

ERROR 2780: Correlated subqueries cannot have more than one level

ERROR 2781: Correlated subqueries with analytics in the select list is not supported

ERROR 2782: Correlated subqueries with no group by and a non-strict expression containing an aggregate in the select list is not supported

ERROR 2783: Correlated subquery column in select/gby/oby not supported

ERROR 2784: Correlated subquery could not be flattened as a join

ERROR 2785: Correlated subquery could not get flattened, a correlated expression could not be treated as a join

ERROR 2786: Correlated subquery expression without aggregates and with limit is not supported

ERROR 2787: Correlated subquery expressions under OR not supported

ERROR 2788: Correlated subquery in expression with operator <> is not supported

ERROR 2790: Correlated subquery with aggregate and limit 0 is not supported

ERROR 2792: Correlated subquery with aggregate function COUNT is not supported

ERROR 2793: Correlated subquery with distinct/group by is not supported

ERROR 2794: Correlated subquery with having clause expression that involves aggregates and subquery is not supported

ERROR 2795: Correlated subquery with NOT IN is not supported

ERROR 2796: Correlated subquery with outer joins and uncorrelated exists is not supported

ERROR 2797: Correlated subquery with User Defined Aggregate is not supported

ERROR 2854: Could not find array type for data type *string*

ERROR 2856: Could not find column *string* in table *string*

ERROR 2942: CREATE ASSERTION is not supported

ERROR 2943: CREATE FUNCTION / INOUT parameters are not supported

ERROR 2944: CREATE FUNCTION / OUT parameters are not supported

ERROR 2980: Data type not supported

ERROR 2981: Data type not supported (*value*)

Vertica Documentation

Vertica Error Messages

ERROR 2983: Database "*string*" does not exist

ERROR 2987: Database references are not supported: "*string.string.string*"

ERROR 3026: Defining query must have a from clause

ERROR 3115: DistinctExpr not supported

ERROR 3116: Distrib overrides are too restrictive. Can not find completed Join Order

ERROR 3118: DML on projection/view is not supported

ERROR 3119: DML query with a predicate that could not be pushed below joins and does not refer solely to the target table is not supported

ERROR 3123: DROP ASSERTION is not supported

ERROR 3126: DROP COLUMN over temporary tables is not supported

ERROR 3132: DROP SEQUENCE does not support CASCADE

ERROR 3141: Dropping local and global objects in one statement is not supported

ERROR 3157: Dynamic load not supported

ERROR 3163: Embedded SQL involving local objects is not supported

ERROR 3174: ENCODED BY is supported in CREATE TABLE ... AS SELECT statement only

ERROR 3246: Error parsing distrib overrides (unexpected end of override); *string*

ERROR 3247: Error parsing distrib value; *string*

ERROR 3291: Event ANY_ROW is not supported

ERROR 3317: Executing when OPT:PLAN_ALL_NODES_ACTIVE option is set

ERROR 3343: Explicit JOIN clause contains a join predicate between relations previously joined

ERROR 3351: Expressions in COPY may not contain aggregate functions

ERROR 3352: Expressions in COPY may not contain analytic or Time Series Aggregate Functions

ERROR 3353: Expressions not supported in Times Series Aggregate Function

ERROR 3357: External tables only support files or a User Defined Source

ERROR 3403: FieldSelect is not supported

ERROR 3404: FieldStore is not supported

ERROR 3417: Final phase output size mismatch

ERROR 3420: First argument of date_part must be a constant string

ERROR 3434: For INSERT SELECT statement, replicated/broadcasted source data not supported

ERROR 3436: For SELECT DISTINCT, ORDER BY expressions must appear in the SELECT clause

ERROR 3451: Function *string* can't be used as a case expression

ERROR 3452: Function *string* can't be used in a boolean

ERROR 3453: Function *string* can't be used in a WHEN clause

ERROR 3454: Function *string* can't be used in as a segment expression

ERROR 3455: Function *string* can't be used with an operator

ERROR 3488: Group By, Order By, Aggregates, Having & limits not allowed in update/delete

ERROR 3510: IGNORE NULLS argument must be a Boolean constant

ERROR 3553: INHERITS not supported

ERROR 3566: Input of anonymous composite types is not implemented

ERROR 3600: Interpolated predicates can accept arguments of the same type only

ERROR 3601: Interpolated predicates can be part of AND expressions only

ERROR 3613: Interval units "*string*" not supported

Vertica Documentation

Vertica Error Messages

ERROR 3821: Joins with an interpolated predicate can have a conjunctive expression containing equality predicates. The equality predicates cannot have expressions or column references with different modifiers

ERROR 3822: Joins with an interpolated predicate cannot have expressions or column references with different modifiers in any of the expressions

ERROR 3857: Library built with unsupported version of Vertica SDK [Version: *string*, Revision: *string*]

ERROR 3876: Locale must be a constant

ERROR 3900: MATCH PARTIAL is not supported

ERROR 3972: Multi-column subquery expressions can only be used with the =, <=> and <> operators

ERROR 3973: Multi-column subquery type ALL can only be used with the = and <=> operators

ERROR 3974: Multi-column subquery type ANY can only be used with the =, <=> and <> operators

ERROR 4106: No single-source bulk loads have been executed in this session

ERROR 4147: Node issuing the query cannot be marked as down

ERROR 4160: Non-equality correlated subquery expression is not supported

ERROR 4170: Not a Star or Snow-Flake Query block

ERROR 4171: Not a Star or Snow-Flake Query block; dimension table not a star or snowflake

ERROR 4172: Not a Star or Snow-Flake Query block; no fact table found

ERROR 4173: Not a Star or Snow-Flake Query block; there are multiple fact tables

ERROR 4197: NULL value found in a column used by a subquery

ERROR 4228: ON COMMIT DROP not supported in CREATE TABLE

ERROR 4238: Only a temporary table projection can be pinned

ERROR 4248: Only inner joins are allowed in the projection defining query

ERROR 4256: Only relations and subqueries are allowed in the FROM clause

ERROR 4258: Only super user can call export_catalog with an output file name

ERROR 4259: Only super user can get the rebalance data script

ERROR 4263: Only superuser can drop system schema

ERROR 4264: Only superuser can rebalance data

ERROR 4265: Only superuser can rebalance data for replicated projections

ERROR 4266: Only superuser can rebalance data for segmented projections

ERROR 4280: Operator *string (value)* is not supported

ERROR 4281: Operator *string* is not supported for row expressions

ERROR 4298: ORDER BY on a UNION/INTERSECT/EXCEPT result must be on one of the result columns

ERROR 4299: ORDER mode not supported

ERROR 4306: OUTER join with broadcasted outer data not supported

ERROR 4307: OUTER or SEMI join - done through CROSS join and FILTER - with replicated outer and segmented inner not supported

ERROR 4308: OUTER relation in OUTER join is not the fact table nor a snowflake dimension table

ERROR 4309: Outer replicated/segmented input to a join cannot be resegmented

ERROR 4310: LEFTOUTER/SEMI/ANTI join with replicated/broadcasted outer data not supported

ERROR 4328: PARTITION AUTO can only be used with single-phase user defined transform functions

ERROR 4329: PARTITION AUTO cannot be used with pattern matching

ERROR 4331: PARTITION BY expression cannot return a tuple

ERROR 4332: PARTITION BY expression has an unknown type

Vertica Documentation

Vertica Error Messages

ERROR 4333: PARTITION BY expression may not contain aggregate functions

ERROR 4335: Partitioning expression not supported for temporary tables

ERROR 4336: Partitioning not supported for temporary tables

ERROR 4352: Pattern "E" is not supported

ERROR 4375: PINNED clause conflicts with KSAFE setting

ERROR 4376: PINNED clause is not supported in CREATE TABLE statement

ERROR 4412: Prepared statements are currently unsupported

ERROR 4465: Projection *string* of local temporary table cannot be created under user schema *string*

ERROR 4471: Projection choices are too restrictive - cannot create correct join between tables

ERROR 4486: Projections are always created and persisted in the default Vertica locale. The current locale is *string*

ERROR 4584: RENAME COLUMN over temporary tables is not supported

ERROR 4586: replicate_catalog has been shut off

ERROR 4628: Row Expressions are not supported in this context

ERROR 4631: ROW syntax is not supported

ERROR 4644: Scalar array expression cannot contain column references or subqueries

ERROR 4645: Scalar array op *string (value)* is not supported

ERROR 4664: Segmentation clause can not have offset in CREATE TABLE statement

ERROR 4665: Segmentation clause with offset conflicts with KSAFE setting

ERROR 4666: Segmentation expression must have integer type

ERROR 4671: SELECT FOR UPDATE cannot be applied to a function

ERROR 4672: SELECT FOR UPDATE cannot be applied to a join

ERROR 4673: SELECT FOR UPDATE cannot be applied to NEW or OLD

ERROR 4674: SELECT FOR UPDATE is not allowed with EXTERNAL TABLES

ERROR 4675: SELECT FOR UPDATE is not allowed with libraries

ERROR 4676: SELECT FOR UPDATE is not allowed with sequences

ERROR 4677: SELECT FOR UPDATE is not allowed with UNION/INTERSECT/EXCEPT

ERROR 4678: SELECT FOR UPDATE is not allowed with views

ERROR 4680: Self joins in UPDATE statements are not allowed

ERROR 4703: Sequence cannot be moved between system schema and user schema

ERROR 4711: Sequence or IDENTITY/AUTO_INCREMENT column in merge query is not supported

ERROR 4714: Sequences are not allowed in default expressions of local temp tables

ERROR 4715: Sequences cannot be called in views

ERROR 4716: Sequences cannot be created under system schemas

ERROR 4728: Set Operator *string* ALL not supported

ERROR 4730: Set Operator queries without a FROM clause are not supported

ERROR 4733: SET SCHEMA over temporary tables is not supported

ERROR 4735: Set-valued function called in context that cannot accept a set

ERROR 4747: SetToDefault is not supported

ERROR 4786: Statement *string* is not supported

ERROR 4808: Subqueries are not supported as the left hand argument to another subquery

ERROR 4809: Subqueries are not supported in the ORDER BY of a timeseries OVER clause

Vertica Documentation

Vertica Error Messages

ERROR 4810: Subqueries are not supported in the ORDER BY of an analytic function OVER clause

ERROR 4812: Subqueries are not supported in the PARTITION BY of an analytic function OVER clause

ERROR 4816: Subqueries in the ON clause are not supported

ERROR 4817: Subqueries in the SELECT or ORDER BY are not supported if the query has aggregates and the subquery is not part of the GROUP BY

ERROR 4818: Subqueries in the SELECT or ORDER BY are not supported if the subquery is not part of the GROUP BY

ERROR 4820: Subqueries in UPDATE/DELETE/MERGE is not supported

ERROR 4821: Subqueries not allowed in target of insert

ERROR 4822: Subqueries referring to no outer columns in HAVING clause when query has aggregates and no GROUP BY are not supported

ERROR 4824: Subquery aggregate expression that refers a correlated column is not supported

ERROR 4839: Subquery type ARRAY is not supported

ERROR 4842: Subquery without a from clause is not supported

ERROR 4850: Support for UPDATE/DELETE/MERGE is not enabled

ERROR 4854: SyncMarkers are not supported

ERROR 4865: System table *string* cannot be created under user schema *string*

ERROR 4869: System view "*string*" cannot be dropped

ERROR 4870: System view *string* cannot be created under user schema *string*

ERROR 4884: Table *string* cannot be created under system schema *string*

ERROR 4897: Table cannot be moved between system schema and user schema

ERROR 4910: Table revalidation error

ERROR 4918: Temporary Sequences are not supported

ERROR 4933: The argument types in a subquery expression in the where/having clause do not match

ERROR 4938: The constant value following the LIMIT clause cannot be negative

ERROR 4939: The constant value following the OFFSET clause cannot be negative

ERROR 4948: The fourth input argument of TIME_SLICE must be START or END

ERROR 4960: The ORDER BY ... USING clause is not supported

ERROR 4966: The second parameter of export_catalog is invalid: *string*

ERROR 4968: The slice length parameter of TIME_SLICE must be a positive integer

ERROR 5005: Time Series Aggregate Function with interpolation scheme LINEAR may only have an INTEGER or FLOAT type as its first argument

ERROR 5016: Time units "*string*" not supported

ERROR 5023: Timeseries output functions are not supported in the ORDER BY of a timeseries OVER clause

ERROR 5028: Timestamp units "*string*" not supported

ERROR 5110: Type *string* (*value*) is not supported

ERROR 5159: Uncorrelated EXISTS subqueries are not supported when the query has both HAVING clause subqueries involving aggregates and when the query has either OUTER JOINS or NOT IN subqueries

ERROR 5160: Uncorrelated EXISTS subqueries in HAVING clause when query has aggregates and no GROUP BY are not supported

ERROR 5195: UNIQUE predicate is not supported

ERROR 5262: Unsafe use of string constant with Unicode escapes

ERROR 5264: Unsupported access to session-scoped (LOCAL) object

ERROR 5270: Unsupported COPY command clause

Vertica Documentation

Vertica Error Messages

ERROR 5275: Unsupported Join in From clause

ERROR 5276: Unsupported Join in From clause: FULL OUTER JOINS not supported

ERROR 5278: Unsupported join of two non-alike segmented projections

ERROR 5280: Unsupported mix of Joins

ERROR 5284: Unsupported query syntax

ERROR 5289: Unsupported subquery expression

ERROR 5291: Unsupported use of aggregates

ERROR 5292: Unsupported use of cursors

ERROR 5293: Unsupported use of DISTINCT clause

ERROR 5294: Unsupported use of FROM clause

ERROR 5295: Unsupported use of GROUP BY or DISTINCT clause

ERROR 5296: Unsupported use of HAVING clause

ERROR 5297: Unsupported use of LIMIT/OFFSET clause

ERROR 5298: Unsupported use of ORDER BY clause

ERROR 5299: Unsupported use of outer joins

ERROR 5300: Unsupported use of query/subquery without FROM clause

ERROR 5301: Unsupported use of sub-queries

ERROR 5302: Unsupported use of target relation

ERROR 5303: Unsupported use of UDF in WHERE clause

ERROR 5304: Unsupported use of UNION/INTERSECT/EXCEPT

ERROR 5313: Update is disallowed on Primary/Foreign Keys columns. Use Delete followed by Insert instead

ERROR 5314: UPDATE may not refer to tables in prejoin projections

ERROR 5366: User defined aggregate cannot be used in query with other distinct aggregates

ERROR 5388: User has insufficient privilege on *string string*

ERROR 5392: User must have the DBDUSER role to run the database designer

ERROR 5396: User projection *string* cannot be created under system schema *string*

ERROR 5402: User-defined transform functions are not supported in the ORDER BY clause

ERROR 5407: VALINDEX column must be the first column in ORDER BY list

ERROR 5426: Vertica currently allows a maximum of *value* physical storage containers per projection

ERROR 5427: Vertica does not support GRANT / REVOKE ON LANGUAGE

ERROR 5428: Vertica does not support GRANT / REVOKE ON TABLESPACE

ERROR 5447: View *string* cannot be created under system schema *string*

ERROR 5456: Volatile functions may not be used in fillers when other computed columns refer to them

ERROR 5465: Window frame exclusion is not supported

ERROR 5530: Audit of external tables is not supported

ERROR 5537: Cannot alter user-defined type "*string*" of column "*string*"

ERROR 5550: COPY from UDSOURCE does not support rejected row numbers with exceptions or rejected data options

ERROR 5551: COPY LOCAL cannot process more than ONE NATIVE or NATIVE VARCHAR file at a time

ERROR 5562: Creating temp tables by LIKE clause is not supported

ERROR 5595: Invalid argument type *string* in function *string*

ERROR 5607: Language of replacement library [*string*] must match language of existing library [*string*]

Vertica Documentation

Vertica Error Messages

ERROR 5681: Unsupported base type *string* for User-defined type *string*

ERROR 5698: Cannot export statistics for the specified object

ERROR 5725: Size specification not supported for User Defined Type *string*

ERROR 5731: The second parameter must be a table/projection/column name

ERROR 5758: Can not drop Filesystem proc *string*

ERROR 5759: Can not drop library "*string*": referenced by storage locations

ERROR 5763: Can't create a managed external table with non-file sources

ERROR 5764: Cannot alter the data type of a table column when a node is down

ERROR 5781: Cannot use meta function or non-deterministic function in SEGMENTED BY expression

ERROR 5859: Due to the data isolation of temp tables with an on-commit-delete-rows policy, the `compute_flexible_keys()` and `compute_flexible_keys_and_build_view()` functions cannot access this table's data

ERROR 5864: Error parsing table (invalid table): *string*

ERROR 5914: HCatalog schema *string* not permitted in search path

ERROR 5990: Projection *string* cannot be created under hcatalog schema *string*

ERROR 5992: Projection cannot be created for HCatalog table *string*

ERROR 6005: Remote table *string.string* found in design query

ERROR 6019: Sequence *string* cannot be created under hcatalog schema *string*

ERROR 6020: Sequence *string* cannot be moved between system schema and hcatalog schema *string*

ERROR 6023: Setting the CPU affinity of the built-in pool "*string*" is not supported

ERROR 6038: Table *string* cannot be created under hcatalog schema *string*

ERROR 6039: Table *string* cannot be moved under hcatalog schema *string*

ERROR 6092: Unsupported access to flex table: No *string* support

ERROR 6108: View *string* cannot be created under hcatalog schema *string*

ERROR 6141: ALTER COLUMN TYPE over tables having aggregate projections is not supported

ERROR 6142: Alter partition not supported for tables with aggregate projections

ERROR 6145: Analytic functions are not allowed in projections

ERROR 6168: Cannot drop column of a table with aggregate projections

ERROR 6173: Cannot open file *string* for debugging ExprHashCode

ERROR 6174: Cannot open/write/close file "*string*" for debugging ExprHashCode

ERROR 6200: Command CREATE INDEX is not supported

ERROR 6233: DISTINCT Aggregates are not allowed in projections

ERROR 6241: DROP COLUMN over tables having aggregate projections is not supported

ERROR 6244: EE option DISABLE_AUTOPARTITION may not be used in conjunction with aggregate projections, or projections with expressions

ERROR 6265: Expressions in the projection SELECT list may not be repeated

ERROR 6266: Expressions on aggregates are not allowed in aggregate projections

ERROR 6322: LIMIT may not be used in projection definitions unless the OVER clause is supplied

ERROR 6323: LIMIT or OVER(...) clause may not be used in projection definitions

ERROR 6332: MERGE is not supported on any target table having projection with expression

ERROR 6380: PARTITION BEST cannot be used with pattern matching

ERROR 6382: PARTITION NODES cannot be used with pattern matching

ERROR 6400: Setting the cascade to pool of the built-in pool "*string*" is not supported

Vertica Documentation

Vertica Error Messages

ERROR 6401: Setting the cascade to pool to the built-in pool "string" is not supported

ERROR 6427: The LIMIT clause does not support OFFSET ... and OVER() clauses simultaneously

ERROR 6455: Unsupported access to table with projection expressions or aggregates

ERROR 6459: UPDATE/DELETE/MERGE a dim table where fact table has aggregate projections is not supported

ERROR 6460: UPDATE/DELETE/MERGE a table with aggregate projections is not supported

ERROR 6464: Use of LIMIT with the OVER(...) clause and DISTINCT is not supported within the same SELECT block

ERROR 6465: Use of LIMIT with the OVER(...) clause and ORDER BY is not supported within the same SELECT block

ERROR 6467: User defined aggregate cannot be used in query with MLAs

ERROR 6476: Cannot directly modify index table constraint

ERROR 6477: Cannot modify index table columns directly

ERROR 6501: Access policy cannot be created on table "string" since it has an aggregate projection

ERROR 6502: Access policy cannot be created on table "string" since it has a pre-join projection

ERROR 6503: Access policy cannot be created on table "string" since it has a projection with expressions

ERROR 6505: Cannot create Aggregate projection on tables with access policy

ERROR 6506: Cannot create prejoin projection on tables with access policy

ERROR 6507: Cannot create projection with expression on tables with access policy

ERROR 6517: Access policy is requested from unsupported object: "string"

ERROR 6522: Cannot alter column width on variable length columns with BLOCKDICTIONARY encoding

ERROR 6525: Direct query to projection "string" are not supported: "string"

ERROR 6530: Invalid statistics file. Total number of bounds specified: 'value' do not match the buckets value: 'value' for column 'string'. Buckets value specified should exactly be same as total number of bounds

ERROR 6531: Invalid table name: 'string'. Make sure the schema/table exists in database

ERROR 6532: Invalid value 'value' for attribute 'string' under bound 'string', column 'string'. Please use a positive value.

ERROR 6533: Invalid value 'value' for attribute 'string' under column 'string'. Please use a positive value.

ERROR 6538: Unable to string: "string"

ERROR 6644: Arguments to Transform Functions in Live Aggregate Projections cannot be constants

ERROR 6670: Cannot alter a table constraint when a node is down

ERROR 6685: Cannot specify anything other than expressions on table columns in the SELECT list

ERROR 6690: Cannot use multi phase UDTs with live aggregate projections

ERROR 6742: drop_location for non-retired DATA locations is not supported

ERROR 6785: For NULLAWARE ANTI JOIN, columns of left-hand-side relation must not appear on the right hand side of ON clause predicates

ERROR 6786: For NULLAWARE ANTI JOIN, columns of right-hand-side relation must appear on the right hand side of ON clause predicates

ERROR 6787: For NULLAWARE ANTI JOIN, columns of right-hand-side relation must not appear on the left hand side of ON clause predicates

ERROR 6817: Input filename cannot be NULL

ERROR 6822: Inputs to batch transform function have to be in the same order as the outputs of the prepass transform function

ERROR 6823: Inputs to batch Transform function may not be expressions

Vertica Documentation

Vertica Error Messages

ERROR 6852: Live Aggregate Projection "*string*" will be created for "*string*". Data in "*string*" will be neither updated nor deleted

ERROR 6862: MARS operation not supported for this client type. Parameter not changed

ERROR 6865: Materialized WITH queries are not supported with directed queries

ERROR 6867: Meta-functions cannot be used in directed queries

ERROR 6882: Null value in ON clause is not supported for Nullaware anti join

ERROR 6888: Only equality join is supported in ON clause of Nullaware anti join

ERROR 6891: Order BY clauses are not allowed in aggregate projection definitions

ERROR 6894: Output filename cannot be NULL

ERROR 6974: System/Virtual tables, projections or views not supported for optimizer-generated annotated queries

ERROR 6983: Tables or projections with access policies not supported for optimizer-generated annotated queries

ERROR 6992: The batch and prepass transform functions must be partitioned by the same values

ERROR 6996: The DB admin has disallowed using the MARS feature

ERROR 7054: Unsupported operation in ON clause of Nullaware anti join

ERROR 7103: Java user defined functions are not supported in aggregate projections

ERROR 7113: Queries with syntactic hints are not supported with directed queries

ERROR 7123: Statement including date/time literals based on current date/time are not supported for directed queries

ERROR 7124: Statement including hints are not supported for directed queries

ERROR 7133: UNI Join with non-subquery inner not supported

ERROR 7169: Constant true/false predicate with ignore constant hint is not supported

ERROR 7188: Object type *value* not supported for access policies

ERROR 7201: The constant value following the LIMIT clause cannot be both zero and ignoreconst

ERROR 7239: Computing flex table keys with non-binary collation locales is not supported

ERROR 7288: Transaction isolation level not supported. Parameter not changed

ERROR 7294: *string* expressions must not return a set

ERROR 7307: Cannot use aggregate functions in *string* expressions

ERROR 7308: Cannot use analytic or time series aggregate functions in *string* expressions

ERROR 7318: Default expressions with subqueries are not supported in a MERGE statement

ERROR 7344: *string* expressions may not refer to other columns with *string* expressions

ERROR 7346: *string* queries may not refer to a temporary table

ERROR 7347: *string* queries must refer to tables

ERROR 7368: ALTER NOTIFIER: Unsupported parameter '*string*'

ERROR 7371: Cannot alter a column's *string* when a node is down

ERROR 7372: Cannot assign value to "*string*" column

ERROR 7394: Columns in COPY may not contain virtual columns

ERROR 7403: describeProjection is not supported in fenced UDX

ERROR 7404: describeTable is not supported in fenced UDX

ERROR 7410: Epsilon must be a non-negative float number and smaller than one-million

ERROR 7421: Expressions on SELECT statements cannot be used in *string* query definitions

ERROR 7422: External tables cannot have *string* expressions

Vertica Documentation

Vertica Error Messages

ERROR 7430: Initial centers in table '*string*' are not distinct

ERROR 7433: Invalid choice for optimizer

ERROR 7439: Invalid type '*string*' for column *string*

ERROR 7440: Invalid type '*string*' for column *string* of initial_centers_table

ERROR 7442: listProjections is not supported in fenced UDX

ERROR 7444: listTableProjections is not supported in fenced UDX

ERROR 7446: listTables is not supported in fenced UDX

ERROR 7450: max_iterations must be a positive integer and less than one-million

ERROR 7455: MD5 can't be used in FIPS mode

ERROR 7466: Must specify either a valid initialization method or initial centers table

ERROR 7475: Not enough number of appropriate samples to run the algorithm (excluding samples with NULL or Inf values)

ERROR 7477: Notifier action is immutable. To set the new action URL you need to drop the notifier, and create a new one

ERROR 7486: num_clusters must be a positive integer

ERROR 7487: Number of clusters must not be bigger than the number of rows without null or infinity values

ERROR 7492: Only 'euclidean' is supported for distance_method

ERROR 7496: Prejoin projection unsupported for table *string* that has default queries or set-using expressions/queries

ERROR 7539: Sequences are not allowed in *string* expressions of local temp tables

ERROR 7548: Tables with prejoin projections cannot have *string* expressions

ERROR 7549: Tables with prejoin projections cannot have *string* queries

ERROR 7550: TableSample does not work with update statements

ERROR 7551: TableSample only works on user defined tables

ERROR 7553: Temporary tables cannot have subqueries as *string* expressions

ERROR 7558: The response_column is included in the predictor_columns. You should exclude it to have a meaningful model

ERROR 7559: The selected normalization method is not supported. Only support MinMax, zscore, robust_zscore at present

ERROR 7560: The selected outlier detection method is not supported. Only support robust_zscore based outlier detection at present

ERROR 7588: Valid value for sampling_ratio is a positive float number between 0 and 1

ERROR 7606: Number of clusters must not be bigger than the number of distinct rows without null or infinity values

ERROR 7631: Response column must be integer or varchar

ERROR 7634: TableSample is not supported with Aggregate projections

ERROR 7635: The selected balance method is not supported. Only support weighted_sampling at present

ERROR 7646: Cursor [*string*] does not support multiple shards

ERROR 7647: Cursor [*string*] does not support multiple storage containers

ERROR 7648: Cursor [*string*] does not support this operation

ERROR 7649: Cursor [*string*] does not support writes

ERROR 7650: describeBlob is not supported in fenced UDX

ERROR 7652: describeFunction does not support function lookup by name / arguments yet

ERROR 7653: describeFunction is not supported in fenced UDX

Vertica Documentation

Vertica Error Messages

ERROR 7654: describeType is not supported in fenced UDX

ERROR 7655: describeType is not supported yet

ERROR 7658: listBlobs is not supported in fenced UDX

ERROR 7659: listBlobs is not supported yet

ERROR 7660: listDerivedTables is not supported in fenced UDX

ERROR 7670: UDX cursors do not currently support deleted data

ERROR 7677: Looking up cursor by table is not yet supported

ERROR 7686: ALTER TABLE does not support ADD CONSTRAINT on text index

ERROR 7687: ALTER TABLE does not support ALTER PARTITION on text index

ERROR 7691: DROP PARTITION is not supported on text index

ERROR 7703: ALTER NOTIFIER: MAXMEMORYSIZE cannot be empty

ERROR 7739: Slices are not supported for VMaps

ERROR 7790: Order BY is not allowed within the OVER() clause of User Defined Transforms in projection definitions

ERROR 7792: Resource pool *string* cannot be modified

ERROR 7815: The column provided in response_column must be of type integer or character

ERROR 7818: Unsupported type '*string*' for column '*string*'

ERROR 7840: Column [*string*] is duplicated in exclude_columns

ERROR 7841: Column [*string*] is duplicated in predictor_columns

ERROR 7846: Relation *string* is empty

ERROR 7848: Table '*string*' contains *value* rows, should be *value*

ERROR 7852: num_clusters must be less than or equal to 1000

ERROR 7867: Live-aggregate projection is not supported for ADD COLUMN ... PROJECTIONS (...)

ERROR 7870: Prejoin projection is not supported for ADD COLUMN ... PROJECTIONS (...)

ERROR 7878: Too many columns in MLA grouping sets: *value*, while only up to *value* MLA columns are allowed

ERROR 7889: ALTER MODEL does not support multiple clauses

ERROR 7898: Model *string* cannot be moved under hcatalog schema *string*

ERROR 7900: Model cannot be moved to system schema

ERROR 7926: Catalog object *string* is not either a table or a schema

ERROR 7927: Column "*string*" is referenced in a column set using expression of projection "*string*" but is not stored in the projection

ERROR 7928: Column list must be empty for refresh_columns on multiple tables

ERROR 7930: COMPLEX JOIN without a boolean marker column not supported

ERROR 7931: COMPLEX JOIN without a complex_join_marker() column at the end of the select list not supported

ERROR 7932: COMPLEX JOIN without a subquery not supported

ERROR 7934: Refresh_columns on multiple tables is only supported for "rebuild" mode

ERROR 7935: Refreshed column cannot be any projection's sort key

ERROR 7936: Refreshed column cannot be referenced in any live-aggregate/expression projection

ERROR 7937: Refreshed column cannot be referenced in projection's segmentation expression

ERROR 7938: Refreshed column cannot be referenced in table's partition expression

ERROR 7964: The selected missing value imputation method is not supported.
Only support mean and mode based missing value imputation at present

ERROR 7974: *string* is not supported in fenced UDX

ERROR 7976: Add column with default subquery is not supported for unsegmented projections

ERROR 7980: Cannot alter model in an incomplete state

ERROR 7991: Keyword "DEFAULT USING" is not supported

ERROR 8006: Qualified column is not supported

ERROR 8008: Refresh_columns with rebuild mode is not supported for unsegmented projections

ERROR 8013: The selected normalization method is not supported. Valid options are 'minmax', 'zscore', and 'robust_zscore'

ERROR 8023: Unsupported column type [*string*] for column [*string*]

ERROR 8029: The column provided in response_column must be of numerical types (INT, FLOAT or NUMERIC)

ERROR 8034: Refresh_columns on enforced PK/Unique constraint columns not supported

ERROR 8038: Cannot alter type of column "*string*" since it is referenced in the SET USING expression of column "*string*"

ERROR 8039: Cannot alter type of column with a SET USING expression

ERROR 8040: Cannot refresh column containing grouped containers

ERROR 8043: Refreshed column cannot be any projection's grouped column

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 0A005

This topic lists the errors associated with the SQLSTATE 0A005.

SQLSTATE 0A005 Description

ERRCODE_PLAN_TO_SQL_NOT_SUPPORTED

Error messages associated with this SQLState

ERROR 6679: Cannot generate annotated query when query contains table that does not have a projection

ERROR 6700: Constant NULLs in NOT-IN clause / Nullaware Anti Join not supported for optimizer-generated annotated queries

ERROR 6739: DML not supported for optimizer-generated annotated queries

ERROR 6770: EXPORT not supported for optimizer-generated annotated queries

ERROR 6866: Meta-function not supported for optimizer-generated annotated queries

ERROR 6889: Optimizer can only generate annotated query for SELECT queries

ERROR 6973: System tables not supported for optimizer-generated annotated queries

ERROR 7114: Query with syntactic hints not supported for optimizer-generated annotated queries

ERROR 7132: Under non-default locale, Multi-level aggregates in set operator sub-queries (except UNION ALL) not supported for optimizer-generated annotated queries

ERROR 7187: Multi-level aggregate queries with more than *value* grouping sets not supported for optimizer-generated annotated queries

ERROR 7449: Match clause not supported for optimizer-generated annotated queries

ERROR 7666: Query with geometry type not supported for optimizer-generated annotated queries

ERROR 7696: Queries with UDX functions with user defined parameter type of *string* is not supported

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE OB000

This topic lists the errors associated with the SQLSTATE OB000.

SQLSTATE OB000 Description

ERRCODE_INVALID_TRANSACTION_INITIATION

Error messages associated with this SQLState

ERROR 2321: Can't start a Transaction in this context

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE OLV01

This topic lists the errors associated with the SQLSTATE OLV01.

SQLSTATE 0LV01 Description

ERRCODE_INVALID_GRANT_OPERATION

Error messages associated with this SQLState

ERROR 2120: Admin option for a role cannot be granted to *string*"public"

ERROR 2601: Circular assignation of roles is not allowed

ERROR 3484: Grant option for a privilege cannot be granted to "public"

ERROR 3485: Grant option for a privilege cannot be granted to (and thus revoked from) "public"

ERROR 3486: Grant options cannot be granted back to your own grantor

ERROR 3616: Invalid *string* statement

ERROR 3719: Invalid option specified for *string* statement

ERROR 3723: Invalid privilege type "*string*"

ERROR 3724: Invalid privilege type *string* for aggregate function

ERROR 3725: Invalid privilege type *string* for analytic function

ERROR 3726: Invalid privilege type *string* for database

ERROR 3727: Invalid privilege type *string* for function

ERROR 3728: Invalid privilege type *string* for library

ERROR 3729: Invalid privilege type *string* for procedure

ERROR 3730: Invalid privilege type *string* for relation

ERROR 3731: Invalid privilege type *string* for resource pool

ERROR 3732: Invalid privilege type *string* for schema

ERROR 3733: Invalid privilege type *string* for sequence

ERROR 3734: Invalid privilege type *string* for storage location

ERROR 3735: Invalid privilege type *string* for transform

ERROR 3745: Invalid role name *string*

ERROR 4056: New *string*

ERROR 4613: Role "*string*" cannot be set as default

ERROR 5601: Invalid privilege type *string* for filter function

ERROR 5602: Invalid privilege type *string* for parser function

ERROR 5603: Invalid privilege type *string* for source function

ERROR 6680: Cannot GRANT *string* authentication to LDAP role

ERROR 6681: Cannot GRANT *string* authentication to LDAP user

ERROR 6682: Cannot GRANT/REVOKE LDAP role to/from LDAP user or role

ERROR 6711: Could not retrieve schema for *string*

ERROR 7163: Cannot materialize schema privileges. Table *string* is not set to include schema privileges

ERROR 7227: Cannot materialize schema privileges. View *string* is not set to include schema privileges

ERROR 7894: Invalid privilege type *string* for model

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22000

This topic lists the errors associated with the SQLSTATE 22000.

SQLSTATE 22000 Description

ERRCODE_DATA_EXCEPTION

Error messages associated with this SQLState

ERROR 3646: Invalid Datum pointer

ERROR 4163: Non-positive value supplied to randomint: *value*

ERROR 4921: Test Error *@string*

ERROR 4922: Test Error from *@string*

ERROR 7474: Non-positive value supplied to randomint_crypto: *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22001

This topic lists the errors associated with the SQLSTATE 22001.

SQLSTATE 22001 Description

ERRCODE_STRING_DATA_RIGHT_TRUNCATION

Error messages associated with this SQLState

ERROR 2991: Date '*string*'*string* too long for type *string*(*value*)

ERROR 3426: Float '*string*'*string* too long for type *string*

ERROR 3589: Integer '*string*'*string* is too long for type *string*(*value*)

ERROR 3605: Interval '*string*'*string* too long for type *string*(*value*)

ERROR 4208: Numeric '*string*' is too long for type *string*

ERROR 4315: Padded octet length (*value*) exceeds the *value* octet limit

ERROR 4604: Result (*value* characters) exceeds the field width (*value*)

ERROR 4800: String of *value* octets is too long for type *string*(*value*)

ERROR 5004: Time '*string*'*string* too long for type *string*(*value*)

ERROR 5024: Timestamp '*string*'*string* too long for type *string*(*value*)

ERROR 5032: Timestamptz '*string*'*string* too long for type *string*(*value*)

ERROR 5035: Timetz '*string*'*string* too long for type *string*(*value*)

ERROR 5417: Value too long for type character varying(*value*)

ERROR 5418: Value too long for type character(*value*)

ERROR 7401: Date '*string*' too long for buffer of length (*value*)

ERROR 7431: Interval '*string*' too long for buffer of length (*value*)

ERROR 7568: Time '*string*' too long for buffer of length (*value*)

ERROR 7569: Timestamp '*string*' too long for buffer of length (*value*)

ERROR 7570: Timestamptz '*string*' too long for buffer of length (*value*)

ERROR 7571: TimeTz '*string*' too long for buffer of length (*value*)

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22003

This topic lists the errors associated with the SQLSTATE 22003.

SQLSTATE 22003 Description

ERRCODE_NUMERIC_VALUE_OUT_OF_RANGE

Error messages associated with this SQLState

ERROR 2429: Cannot find matching query in the system

ERROR 2828: Could not convert '*string*'*string* to an int8

ERROR 3425: Float "*value*" is out of range for type *string*

ERROR 3675: Invalid input for *string*, exceeds 32 bits: "*string*"

ERROR 3676: Invalid input for *string*, exceeds 64 bits: "*string*"

ERROR 3786: Invalid value for float: "*string*"

ERROR 4200: Number of buckets must be a positive integer

ERROR 4361: Percentile value must be a number between 0 and 1

ERROR 4704: Sequence exceeded max value

ERROR 4705: Sequence exceeded min value

ERROR 4756: Smoothing factor must between 0 and 1

ERROR 4795: String "*string*"*string* is out of range as a float8

ERROR 4796: String "*string*"*string* is out of range as an int8

ERROR 4845: Sum() overflowed

ERROR 5408: Value "*value*" is out of range for type *string*

ERROR 5409: Value "*string*" is out of range for type int8

ERROR 5411: Value exceeds range of type *string*

ERROR 5412: Value is too long for type *string*: "*value*"

ERROR 6063: Total number of significant digits for value *string* is more than what is defined. Buffer size is *value* while actual length of word is *value* instead

ERROR 7623: Value "*string*" is out of range for type int64

ERROR 7697: Arithmetic overflow accumulating numeric, operand %Lg

ERROR 7698: Arithmetic overflow adding numerics, operands %Lg, %Lg

ERROR 7699: Arithmetic overflow subtracting numerics, operands %Lg, %Lg

ERROR 7986: Evaluation of expression to be inserted exceeded range of type numeric(*value*,*value*)

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22004

This topic lists the errors associated with the SQLSTATE 22004.

SQLSTATE 22004 Description

ERRCODE_NULL_VALUE_NOT_ALLOWED

Error messages associated with this SQLState

ERROR 2110: ACL arrays must not contain null values

ERROR 2501: Cannot set a NOT NULL column (*value*) to a NULL value in *value* statement

ERROR 2502: Cannot set a NOT NULL column (*string*) to a NULL value in INSERT/UPDATE statement

ERROR 2514: Cannot set NOT NULL columns (*string*) to a NULL value in INSERT/UPDATE statement

ERROR 4195: NULL value detected in data partitioning expression

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22007

This topic lists the errors associated with the SQLSTATE 22007.

SQLSTATE 22007 Description

ERRCODE_INVALID_DATETIME_FORMAT

Error messages associated with this SQLState

ERROR 2171: AM/PM hour (*value*) must be between 1 and 12

ERROR 2364: Cannot calculate day of year without year information

ERROR 3439: Format *string* is invalid for an Interval value

ERROR 3535: Inconsistent use of year *value* and "BC"

ERROR 3647: Invalid day-of-week '*string*'

ERROR 3679: Invalid input syntax for *string*: "*string*"

ERROR 3721: Invalid partition key

ERROR 3785: Invalid value for *string*: "*string*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22008

This topic lists the errors associated with the SQLSTATE 22008.

SQLSTATE 22008 Description

ERRCODE_DATETIME_FIELD_OVERFLOW

Error messages associated with this SQLState

ERROR 2992: Date/time field value out of range: "*string*"

ERROR 4065: next_day(infinity, DOW) is not defined

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22009

This topic lists the errors associated with the SQLSTATE 22009.

SQLSTATE 22009 Description

ERRCODE_INVALID_TIME_ZONE_DISPLACEMENT_VALUE

Error messages associated with this SQLState

ERROR 3768: Invalid timezone interval displacement

ERROR 5044: Timezone displacement out of range: "string"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 2200B

This topic lists the errors associated with the SQLSTATE 2200B.

SQLSTATE 2200B Description

ERRCODE_ESCAPE_CHARACTER_CONFLICT

Error messages associated with this SQLState

ERROR 2699: Conflicting or redundant options

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 2200D

This topic lists the errors associated with the SQLSTATE 2200D.

SQLSTATE 2200D Description

ERRCODE_INVALID_ESCAPE_OCTET

Error messages associated with this SQLState

ERROR 3285: ESCAPE strings must be a single octet, not "value"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22011

This topic lists the errors associated with the SQLSTATE 22011.

SQLSTATE 22011 Description

ERRCODE_SUBSTRING_ERROR

Error messages associated with this SQLState

ERROR 4034: Negative count not allowed

ERROR 4035: Negative length not allowed

ERROR 4036: Negative or zero substring start position not allowed

ERROR 4039: Negative substring length not allowed

ERROR 4784: Start position cannot be 0

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22012

This topic lists the errors associated with the SQLSTATE 22012.

SQLSTATE 22012 Description

ERRCODE_DIVISION_BY_ZERO

Error messages associated with this SQLState

ERROR 3117: Division by zero

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22015

This topic lists the errors associated with the SQLSTATE 22015.

SQLSTATE 22015 Description

ERRCODE_INTERVAL_FIELD_OVERFLOW

Error messages associated with this SQLState

ERROR 3606: Interval field value out of range: "*string*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22019

This topic lists the errors associated with the SQLSTATE 22019.

SQLSTATE 22019 Description

ERRCODE_INVALID_ESCAPE_CHARACTER

Error messages associated with this SQLState

ERROR 2729: COPY DELIMITER for column *string* must be a single character
ERROR 2730: COPY delimiter must be a single character
ERROR 2731: COPY ENCLOSED BY cannot be a whitespace character
ERROR 2732: COPY ENCLOSED BY for column *string* cannot be a whitespace character
ERROR 2733: COPY ENCLOSED BY for column *string* must be a single character
ERROR 2734: COPY ENCLOSED BY must be a single character
ERROR 2736: COPY ESCAPE AS for column *string* must be a single character
ERROR 2737: COPY ESCAPE must be a single character
ERROR 2758: COPY TRIM for column *string* must be an empty string or a single character
ERROR 2759: COPY trim must be an empty string or a single character
ERROR 3284: ESCAPE strings must be a single character, not "*value*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 2201B

This topic lists the errors associated with the SQLSTATE 2201B.

SQLSTATE 2201B Description

ERRCODE_INVALID_REGULAR_EXPRESSION

Error messages associated with this SQLState

ERROR 3742: Invalid regexp match_param: '*character*'
ERROR 4552: Regexp match or recursion limit exceeded (*rc value*)
ERROR 4553: Regexp pattern error at offset *value*: *string*
ERROR 4554: Regexp pattern study error: *string*
ERROR 5064: Too many regular expression subexpressions

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 2201G

This topic lists the errors associated with the SQLSTATE 2201G.

SQLSTATE 2201G Description

ERRCODE_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION

Error messages associated with this SQLState

ERROR 2939: Count must be greater than zero

ERROR 3888: Lower and upper bounds must be finite

ERROR 3889: Lower bound cannot equal upper bound

ERROR 4277: Operand, lower bound and upper bound cannot be NaN

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22021

This topic lists the errors associated with the SQLSTATE 22021.

SQLSTATE 22021 Description

ERRCODE_CHARACTER_NOT_IN_REPERTOIRE

Error messages associated with this SQLState

ERROR 4551: Regexp encountered an invalid UTF-8 character

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22023

This topic lists the errors associated with the SQLSTATE 22023.

SQLSTATE 22023 Description

ERRCODE_INVALID_PARAMETER_VALUE

Error messages associated with this SQLState

ERROR 2007: *string* can not be greater than PASSWORD_MAX_LENGTH *value*

ERROR 2008: *string* can not be set to a negative number

ERROR 2028: *string* exceptions and rejected_data can not be the same filename

ERROR 2033: *string* input file and exceptions can not be the same filename

ERROR 2034: *string* input file and rejected_data can not be the same filename

ERROR 2042: *string* must be a positive integer

ERROR 2048: *string* Path [*string*] is a directory

ERROR 2049: *string* Path [*string*] is a socket

ERROR 2051: *string* Record terminator length (*value*) is larger than load read buffer size (*value*)

ERROR 2056: *string* Unrecognized format '*string*' for column *value*

ERROR 2071: '*string*' is not a valid size description

ERROR 2075: @INCLUDE without filename in timezone file "*string*", line *value*

ERROR 2077: [*string*] cannot be dropped. There will be no storage locations for data files

ERROR 2078: [*string*] cannot be dropped. There will be no storage locations for temporary files

ERROR 2079: [*string*] cannot be retired. There will be no storage locations for data files

ERROR 2080: [*string*] cannot be retired. There will be no storage locations for temporary files

ERROR 2081: [*string*] is not a valid storage location on node *string*

Vertica Documentation

Vertica Error Messages

ERROR 2108: ACL array contains wrong data type

ERROR 2109: ACL arrays must be one-dimensional

ERROR 2158: All columns of soft unique key statistics must be from the same table

ERROR 2194: analyze_statistics: Can not analyze statistics of a non-local temporary table/projection '*string*'

ERROR 2195: analyze_statistics: Can not analyze statistics of a virtual table/projection *string*

ERROR 2196: analyze_statistics: Cannot analyze statistics of a virtual table *string*

ERROR 2197: analyze_statistics: invalid accuracy *value* A number between 0 and 100 is required

ERROR 2254: Bad snapshot name '*string*' (cannot contain / or start with a .)

ERROR 2298: Can not lock/unlock super user account

ERROR 2300: Can not reuse any recent passwords

ERROR 2301: Can not reuse current password

ERROR 2302: Can not reuse the previous *value* passwords

ERROR 2317: Can't purge projection(s); AHM is at epoch 0

ERROR 2319: Can't set a REJECTED file on node '*string*', which the current query is not executing on

ERROR 2320: Can't set an EXCEPTIONS file on node '*string*', which the current query is not executing on

ERROR 2365: Cannot calculate week number without year information

ERROR 2370: Cannot close a protected session

ERROR 2414: Cannot drop extended statistics on a projection (*string*)

ERROR 2415: Cannot drop extended statistics on projection *string*. Dropping base statistics only

ERROR 2452: Cannot load data from node *string* as it is down

ERROR 2457: Cannot merge partitions in multiple tables at the same time

ERROR 2468: Cannot partition by value multiple tables at the same time

ERROR 2478: Cannot release savepoint; no transaction in progress

ERROR 2500: Cannot set *string* maxConcurrency to unlimited

ERROR 2508: Cannot set maxConcurrency of *string* pool to 0

ERROR 2509: Cannot set maxMemorySize of *string* pool to *string* [*value* KB], as it is above 75% [75% = *value* KB]

ERROR 2510: Cannot set maxMemorySize of *string* pool to none, as this could prevent moveout from running

ERROR 2511: Cannot set maxMemorySize of recovery pool to *string* [*value* KB], as it is below 25% [*value* KB]

ERROR 2513: Cannot set memorySize of general pool

ERROR 2523: Cannot specify exceptions or rejected-data files ON ANY NODE

ERROR 2540: Cannot use 0 for a key, used internally

ERROR 2548: Cannot use both COPY LOCAL and ON ANY NODE: LOCAL files are stored on the client, not on any Vertica node

ERROR 2621: Collection type must be specified

ERROR 2624: Column "*string*" does not exist

ERROR 2653: Column *string* of projection *string* has ACCESSRANK < 0

ERROR 2695: Conflicting "datestyle" keywords

ERROR 2720: Conversion to timezone "*string*" failed

ERROR 2722: COPY .. LOCAL cannot store *string* on a Vertica node

ERROR 2723: COPY ... LOCAL can read files from the client only

Vertica Documentation

Vertica Error Messages

ERROR 2724: COPY ... LOCAL can read files with same compression only

ERROR 2727: COPY column option *string* not supported with format *string*

ERROR 2728: COPY delimiter *string* must not appear in the NULL specification

ERROR 2735: COPY ENCLOSING CHARACTER *string* must not appear in the NULL specification

ERROR 2748: COPY NULL must be an empty string or a single character for FIXED WIDTH data

ERROR 2749: COPY option *string* not supported

ERROR 2750: COPY option *string* not supported with format *string*

ERROR 2752: COPY RECORD TERMINATOR must be at least ONE character long

ERROR 2753: COPY REJECTMAX should be ≥ 0

ERROR 2756: COPY skip characters should be ≥ 0

ERROR 2757: COPY skip should be ≥ 0

ERROR 2760: COPY WITH PARSER Error (column *value*): Parser specified a column of type [*string*]; table needs [*string*]

ERROR 2761: COPY WITH PARSER Error: Parser specified *value* column(s); table needs *value* column(s)

ERROR 2765: COPY: width and length of null string does not match for column *string*

ERROR 2766: COPY: width for column *string* has to be greater than 0

ERROR 2830: Could not convert to timezone "*string*"

ERROR 2932: Couldn't find the specified task

ERROR 2950: Current design does not meet the requirements for K = *value*
Current design is valid for K *string value*
string

ERROR 2963: CURRENT_TIME(*value*) precision must not be negative

ERROR 2964: CURRENT_TIME(*value*) precision reduced to maximum allowed, *value*

ERROR 2965: CURRENT_TIMESTAMP(*value*) precision must not be negative

ERROR 2966: CURRENT_TIMESTAMP(*value*) precision reduced to maximum allowed, *value*

ERROR 2993: Datepart "*string*" not recognized

ERROR 2994: Datepart is invalid

ERROR 3006: DDL statement interfered with snapshot; an object no longer exists

ERROR 3007: DDL statement interfered with this statement

ERROR 3012: DECIMAL precision *value* must be between 1 and *value*

ERROR 3013: DECIMAL scale *value* must be between 0 and precision *value*

ERROR 3032: Delimiter and record terminator cannot be the same value

ERROR 3033: Delimiter and record terminator for *string* cannot be the same value

ERROR 3137: drop_statistics: Can not drop base or histogram statistics of a non-local temporary table/projection *string*

ERROR 3138: drop_statistics: Can not drop statistics for a virtual table/projection *string*

ERROR 3139: drop_statistics: Invalid stats type '*string*'. Valid values are 'base', 'histograms' and 'extended'

ERROR 3168: ENCLOSED BY and delimiter *string* can not be the same value

ERROR 3169: ENCLOSED BY and ESCAPE AS *string* can not be the same value

ERROR 3170: ENCLOSED BY and record terminator *string* can not be the same value

ERROR 3178: ENFORCELENGTH cannot be specified for *string*

ERROR 3280: ESCAPE AS and delimiter *string* can not be the same value

Vertica Documentation

Vertica Error Messages

ERROR 3281: ESCAPE AS and NULL specification *string* can not be the same value

ERROR 3282: ESCAPE AS and record terminator *string* can not be the same value

ERROR 3383: Failed to parse object name string

ERROR 3423: Fixed width record size (*value*) is too large. Record size has to be lesser than *value* (*0xvalue*)

ERROR 3424: Fixed width record size is too large. Record size has to be lesser than *value* (*0xvalue*)

ERROR 3440: Format cannot be specified for *string*

ERROR 3503: ICU *string* error: '*string*'

ERROR 3505: ICU does not support locale '*string*'

ERROR 3513: Illegal argument to `change_runtime_priority`: NULL

ERROR 3514: Illegal argument to `set_config_parameter`: NULL

ERROR 3524: In the SAMPLE STORAGE *n* or SAMPLE STORAGE *n*,*b* clause, *n* must be a constant greater than or equal to 0

ERROR 3525: In the SAMPLE STORAGE *n* PERCENT or SAMPLE STORAGE *n* PERCENT,*b* clause, *n* must be a constant greater than or equal to 0 and less than or equal to 100

ERROR 3526: In the SAMPLE STORAGE *n* PERCENT,*b* clause, *n* must be a constant greater than or equal to 0 and less than or equal to 100, while *b* must be a constant greater than or equal to 0

ERROR 3527: In the SAMPLE STORAGE *n*,*b* clause, both *n* and *b* must be constants greater than or equal to 0

ERROR 3528: In the SAMPLE STORAGE *n*,*b* or SAMPLE STORAGE *n* PERCENT,*b* clause, *b* must be a constant greater than or equal to 0

ERROR 3541: Increase in pool size to *string* [*value* KB] causes general pool to fall below minimum [25% = *value* KB]

ERROR 3607: INTERVAL leading field precision increased to *value*

ERROR 3608: INTERVAL leading field precision reduced to *value*

ERROR 3610: INTERVAL SECOND precision reduced to *value*

ERROR 3612: Interval units "*value*" not recognized

ERROR 3618: Invalid accuracy value for `analyze_histogram`

ERROR 3632: Invalid collection type *string* specified

ERROR 3652: Invalid Directives type: *string*

ERROR 3673: Invalid hint identifier '*string*'

ERROR 3686: Invalid interval value for `timezone`

ERROR 3688: Invalid K value: *value* K cannot be less than zero

ERROR 3689: Invalid K value: *value* Maximum K value for *value* nodes is: *value*

ERROR 3692: Invalid limit type (*string*): must be HIGH or LOW

ERROR 3695: Invalid list syntax for "`datestyle`"

ERROR 3707: Invalid node: [*string*]

ERROR 3710: Invalid number for `timezone` offset in `timezone` file "*string*", line *value*

ERROR 3741: Invalid range

ERROR 3743: Invalid resource type (*string*)

ERROR 3746: Invalid runtime priority string

ERROR 3750: Invalid service name for '*string*'

ERROR 3759: Invalid syntax in `timezone` file "*string*", line *value*

ERROR 3767: Invalid `timezone` file name "*string*"

ERROR 3777: Invalid Usage type: *string*

Vertica Documentation

Vertica Error Messages

ERROR 3780: Invalid user/role name "*string*"

ERROR 3783: Invalid value *string=string*

ERROR 3787: Invalid value for parameter

ERROR 3788: Invalid value for parameter *string: string*

ERROR 3789: Invalid value for search path: "*string*"

ERROR 3840: Keyword '*string*' (*string=string*) is not supported

ERROR 3845: Latency should be > 0

ERROR 3852: Length for type *string* cannot exceed *value*

ERROR 3853: Length for type *string* must be at least 1

ERROR 3877: LOCALTIME(*value*) precision must not be negative

ERROR 3878: LOCALTIME(*value*) precision reduced to maximum allowed, *value*

ERROR 3879: LOCALTIMESTAMP(*value*) precision must not be negative

ERROR 3880: LOCALTIMESTAMP(*value*) precision reduced to maximum allowed, *value*

ERROR 3912: maxMemorySize of *string* [*value* KB] is not in bounds [max is *value* KB]

ERROR 3920: memoryCap of *string* (*value* KB) would exceed [*value* KB]

ERROR 3922: memorySize *string* [*value* KB] would exceed maxMemorySize *string* [*value* KB]

ERROR 3923: memorySize of *string* [*value* KB] would exceed [*value* KB]

ERROR 3960: Missing timezone abbreviation in timezone file "*string*", line *value*

ERROR 3961: Missing timezone offset in timezone file "*string*", line *value*

ERROR 3967: More than one *string* specified for a node

ERROR 4027: Must supply a CATALOGPATH

ERROR 4028: Must supply a HOSTNAME

ERROR 4037: Negative run time cap is not allowed

ERROR 4038: Negative runTimeCap is not allowed

ERROR 4089: No objects specified

ERROR 4175: Not allowed to close session

ERROR 4186: NULL is an invalid K value

ERROR 4187: NULL is invalid object name for analyze_extended_statistics

ERROR 4188: NULL is invalid object name for analyze_histogram

ERROR 4189: NULL is invalid object name for drop_statistics

ERROR 4190: NULL is invalid scope type for analyze_extended_statistics

ERROR 4191: NULL is invalid statistics type for analyze_extended_statistics

ERROR 4192: NULL is invalid statistics type for drop_statistics

ERROR 4194: NULL string and record terminator *string* can not be the same value

ERROR 4211: NUMERIC precision *value* must be between 1 and *value*

ERROR 4212: NUMERIC scale *value* must be between 0 and precision *value*

ERROR 4222: Occurrence number must be > 0

ERROR 4250: Only ONE exception file should be specified for a LOCAL copy

ERROR 4252: Only ONE rejected data file should be specified for a LOCAL copy

ERROR 4318: Parameter *string* in default profile can not be set to DEFAULT

ERROR 4319: Parameter *string* may not exceed 9999

ERROR 4330: PARTITION BY clause must contain table columns in a valid expression

Vertica Documentation

Vertica Error Messages

ERROR 4334: Partition key too long

ERROR 4344: PASSWORD_MAX_LENGTH must be within the range from *value* to *value*

ERROR 4345: PASSWORD_MIN_DIGITS + PASSWORD_MIN_SYMBOLS + PASSWORD_MIN_LETTERS *value* can not be greater than PASSWORD_MAX_LENGTH *value*

ERROR 4346: PASSWORD_MIN_DIGITS + PASSWORD_MIN_SYMBOLS + PASSWORD_MIN_LOWERCASE_LETTERS + PASSWORD_MIN_UPPERCASE_LETTERS *value* can not be greater than PASSWORD_MAX_LENGTH *value*

ERROR 4347: Path cannot be an empty string

ERROR 4406: Precision for type float must be at least 1 bit

ERROR 4407: Precision for type float must be less than 54 bits

ERROR 4408: Precision must be less than *value*; result would be numeric(*value,value*)

ERROR 4454: Projection *string* cannot be analyzed, because it is not up to date

ERROR 4456: Projection *string* cannot drop statistics, because it is not up to date

ERROR 4529: Rebalance skew percent must be in the range [0,100]

ERROR 4556: Regexp starting position must be greater than zero

ERROR 4595: Resource pool "*string*" is an internal pool and cannot be dropped

ERROR 4606: Retention settings must be less than 2TB

ERROR 4639: Run time cap cannot exceed 1 year

ERROR 4642: runTimeCap cannot exceed 1 year

ERROR 4647: Scaling factor must be greater than zero

ERROR 4648: Scaling factor must be less than 33

ERROR 4653: Schema *string* is virtual

ERROR 4701: Sequence *string* is already owned by *string*

ERROR 4702: SEQUENCE CACHE should be greater than 0

ERROR 4708: SEQUENCE MAXVALUE is too large and will overflow

ERROR 4709: SEQUENCE MINVALUE is too small and will underflow

ERROR 4710: SEQUENCE MINVALUE should be lesser than MAXVALUE

ERROR 4712: SEQUENCE START WITH should be between MINVALUE and MAXVALUE

ERROR 4723: SET *string* takes only one argument

ERROR 4745: Setting sysdata maxMemorySize below 4 MB to *string* [*value* KB] will prevent system table queries from running

ERROR 4766: Specified too few widths for the given number of columns

ERROR 4770: Specify at least one table-column for soft unique key statistics

ERROR 4802: STROKE collations are not supported

ERROR 4807: Subnet mask is empty

ERROR 4862: System pool priority must be between -110 and 110 inclusive

ERROR 4889: Table *string* is already owned by *string*

ERROR 4892: Table *string* is not partitioned

ERROR 4893: Table *string* is session scoped

ERROR 4894: Table *string* is virtual

ERROR 4923: That password is not acceptable

ERROR 4937: The confidence level must be between 0 and 100 inclusive.
string

Vertica Documentation

Vertica Error Messages

ERROR 4961: The permissible error must be between 0 and 100 inclusive.
string

ERROR 4985: There is no reason to set *string.string*. Consult documentation

ERROR 5002: Throughput should be > 0

ERROR 5014: Time units "*value*" not recognized

ERROR 5015: Time units "*string*" not recognized

ERROR 5019: TIME(*value*)*string* precision must not be negative

ERROR 5020: TIME(*value*)*string* precision reduced to maximum allowed, *value*

ERROR 5026: Timestamp units "*value*" not recognized

ERROR 5027: Timestamp units "*string*" not recognized

ERROR 5029: TIMESTAMP(*value*) precision reduced to maximum allowed, *value*

ERROR 5030: TIMESTAMP(*value*)*string* precision must not be negative

ERROR 5031: TIMESTAMP(*value*)*string* precision reduced to maximum allowed, *value*

ERROR 5034: TIMESTAMPTZ(*value*) precision must not be negative

ERROR 5036: TIMETZ(*value*) precision must not be negative

ERROR 5037: TIMETZ(*value*) precision reduced to maximum allowed, *value*

ERROR 5038: Timezone "*string*" not recognized

ERROR 5039: Timezone "*string*" uses leap seconds

ERROR 5041: Timezone abbreviation "*string*" is multiply defined

ERROR 5042: Timezone abbreviation "*string*" is too long (maximum *value* characters) in timezone file "*string*", line *value*

ERROR 5045: Timezone file recursion limit exceeded in file "*string*"

ERROR 5046: Timezone offset *value* is not a multiple of 900 sec (15 min) in timezone file "*string*", line *value*

ERROR 5047: Timezone offset *value* is out of range in timezone file "*string*", line *value*

ERROR 5048: Timezone value "*string*" is more than *value* hours

ERROR 5067: Total data collector memory retention of *value*KB is too large given system memory size

ERROR 5106: TuningRecommendations data collection is disabled

ERROR 5118: UDL specified no execution nodes; at least one execution node must be specified

ERROR 5136: Unable to log this tuning analysis event

ERROR 5198: Unknown authentication method: "*string*"

ERROR 5209: Unknown node: *string*

ERROR 5211: Unknown or unsupported object: *string*

ERROR 5215: Unknown value *string*=*string*

ERROR 5220: Unrecognized "datestyle" keyword: "*string*"

ERROR 5229: Unrecognized format '*string*'

ERROR 5248: Unrecognized privilege type: "*string*"

ERROR 5258: Unrecognized timezone name: "*string*"

ERROR 5271: Unsupported format code: *value*

ERROR 5316: Usage cannot be an empty string

ERROR 5317: Usage of [*string*] cannot be changed from *string* to *string*

Vertica Documentation

Vertica Error Messages

ERROR 5319: Usage of [*string*] cannot be changed to *string*. There will be no storage locations for data files

ERROR 5320: Usage of [*string*] cannot be changed to *string*. There will be no storage locations for temporary files

ERROR 5322: Usage:

ERROR 5393: User pool priority must be between -100 and 100 inclusive

ERROR 5437: Vertica should not be run with less than 1GB of RAM

ERROR 5520: *string* compresses network traffic. *string* does NOT compress network traffic. Please change the configuration to be consistent

ERROR 5521: *string* does NOT compress network traffic. *string* compresses network traffic. Please change the configuration to be consistent

ERROR 5538: Cannot COPY user-defined types directly. Please compute them using copy expressions

ERROR 5542: Cannot INSERT or COPY user-defined types directly. Please compute them using appropriate user-defined functions

ERROR 5545: Cluster layout must include all non-ephemeral nodes and should also not include any ephemeral nodes

ERROR 5549: Conversion from *string* to DataType *string* failed. Invalid value

ERROR 5571: Empty storage tier label is not allowed

ERROR 5576: Every permanent node should only be listed once

ERROR 5598: Invalid or unavailable type 'LONG VARBINARY'

ERROR 5599: Invalid or unavailable type 'LONG VARCHAR'

ERROR 5605: Invalid projection createtype '*string*'

ERROR 5613: Length for type *string* must be between 1 and *value*

ERROR 5631: Object *string* does not exist or is not of supported type

ERROR 5632: Object *string* is not a table

ERROR 5634: Path [*string*] is a directory

ERROR 5644: Projection basename "*string*" is not a prefix of projection name "*string*"

ERROR 5645: Projection basename cannot be empty

ERROR 5646: Projection createtype cannot be empty

ERROR 5647: Provided Node "*string*" does not exist

ERROR 5648: Provided Node "*string*" is not permanent

ERROR 5668: Target table name can not be empty

ERROR 5685: User Defined Filter expected but found *string*

ERROR 5686: User Defined Parser expected but found *string*

ERROR 5687: User Defined Source expected but found *string*

ERROR 5693: Using 1 year for QUEUETIMEOUT

ERROR 5703: Couldn't find the specified task, or the Resource Manager has not recieved the request

ERROR 5728: Specified too many widths (*value*) for the given number of columns (*value*)

ERROR 5740: '*string*' is not a valid value for database option *string*

ERROR 5746: analyze_statistics: invalid number of buckets *value*. A number > 0 is required

ERROR 5750: Attempt to configure CPU affinity mode conflicts with configuration of resource pool '*string*'

ERROR 5751: Attempt to configure CPU affinity set to '*string*' conflicts with configuration of resource pool '*string*'

Vertica Documentation

Vertica Error Messages

ERROR 5752: Attempt to configure CPU affinity set to 'string' in exclusive mode would not leave any CPUs available for system queries

ERROR 5753: Attempt to configure CPU affinity to exclusive mode would not leave any CPUs available for system queries

ERROR 5762: Can only specify user defined file system for DATA and/or TEMP storage locations

ERROR 5767: Cannot do LOCAL and REJECTED DATA AS TABLE in the same query; rejected records can only be saved to one location

ERROR 5768: Cannot do RETURNREJECTED and REJECTED DATA AS TABLE in the same query; rejected records can only be saved to one location

ERROR 5778: Cannot specify both a rejected file and a rejected table in the same statement

ERROR 5779: Cannot specify both an exceptions file and a rejected table in the same statement

ERROR 5793: Control set size out of bounds $-1 \leq value \leq 128$

ERROR 5804: CPU #value is not available to this server, because of server-level processor pinning

ERROR 5918: Improperly formatted broadcast address [string]

ERROR 5925: Interface IP address (family string) "string" is invalid

ERROR 5933: Invalid state for UDFilter: REJECT

ERROR 5934: Invalid state for UDSOURCE: INPUT_NEEDED

ERROR 5935: Invalid state for UDSOURCE: REJECT

ERROR 5954: memoryCap of string [value KB] would exceed [value KB]

ERROR 5962: Must request a positive key count to materialize: value

ERROR 5976: Object already exists: string. Can't create a rejections table with the same name

ERROR 6006: Request for value percent of value CPUs rounds to zero CPUs

ERROR 6007: Request for reservation of CPU #value conflicts with another pool's reservation

ERROR 6010: Resource pool 'string' not found

ERROR 6044: Table already exists: string. Can't create a rejections table with the same name

ERROR 6045: The CPU affinity mode cannot be SHARED or EXCLUSIVE if the affinity set is empty

ERROR 6073: Unable to allocate value CPUs for resource pool in string affinity mode

ERROR 6088: Unknown control mode string

ERROR 6090: Unknown database option 'string'

ERROR 6097: User-Defined Load function indicated that it consumed value bytes, when only value were available

ERROR 6123: A Resource Pool cannot cascade to itself

ERROR 6126: Access policy cannot be altered on unknown table "string"

ERROR 6127: Access policy cannot be copied from unknown object "string"

ERROR 6128: Access policy cannot be copied to unknown object "string"

ERROR 6129: Access policy cannot be created on temporary table "string"

ERROR 6130: Access policy cannot be created on unknown table "string"

ERROR 6131: Access policy cannot be created with empty expression

ERROR 6132: Access policy cannot be dropped on unknown object "string"

ERROR 6134: Access policy for COLUMN "string" already exists on "string"

ERROR 6135: Access policy for ROWS already exists on "string"

ERROR 6150: Can not alter access policy on table "string" for COLUMN "string": it doesn't exist

ERROR 6151: Can not alter access policy on table "string" for ROWS: it doesn't exist

Vertica Documentation

Vertica Error Messages

ERROR 6152: Can not copy access policy from table "*string*" for COLUMN "*string*": it doesn't exist

ERROR 6153: Can not copy access policy from table "*string*" for ROWS: it doesn't exist

ERROR 6154: Can not drop access policy on table "*string*" for COLUMN "*string*": it doesn't exist

ERROR 6155: Can not drop access policy on table "*string*" for ROWS: it doesn't exist

ERROR 6159: Can't drop indexed column

ERROR 6160: Can't *string* between these two tables: Text indices don't line up

ERROR 6161: Can't specify a schema for LOCAL TEMP objects

ERROR 6167: Cannot directly modify a text-index table

ERROR 6170: Cannot index external table '*stringstringstring*'

ERROR 6172: Cannot move data from a non retired storage location on node *string*

ERROR 6175: Cannot perform the specified administrative operation on a table with a text index

ERROR 6178: Cannot resolve node address [*string*] using address family *string*

ERROR 6179: Cannot resolve node address [*string*] using family *string*

ERROR 6180: Cannot resolve node control address [*string*] using address family *string*

ERROR 6181: Cannot resolve node control address [*string*] using family *string*

ERROR 6189: Client Authentication "*string*" is disabled

ERROR 6194: Column "*string*" doesnt exists in "*string*". Cannot create access policy

ERROR 6202: Correlation STRENGTH must be in the range [-1.0,1.0]

ERROR 6203: Could not access *string* "*string*": *value*

ERROR 6275: General pool priority must be between -100 and 100 inclusive

ERROR 6283: Illegal argument to get_config_parameter: NULL

ERROR 6288: Indexed table must have a projection that is segmented by HASH(*string*), sorted by *string*, and contains every column listed in the CREATE TEXT INDEX statement

ERROR 6289: Indexed table must have a projection that is segmented by HASH(*string*), sorted by *string*, and contains every column listed in the CREATE TEXT INDEX statement (no projections have been created yet)

ERROR 6295: Invalid declaration; 'tls' method requires HOST TLS

ERROR 6296: Invalid ip address and/or network mask - "*string*"

ERROR 6297: Invalid number of control nodes *value*

ERROR 6298: Invalid parameter - "*string* = *string*"

ERROR 6299: Invalid resource pool specified for cascade to

ERROR 6302: IP address family conflict: node address family is *string*, control address family is *string*

ERROR 6314: LDAP URL [*string*] seems to be malformed*string*. Client Authentication "*string*" is disabled

ERROR 6329: Malformed message description

ERROR 6368: Only simple "expression" is allowed in access policy

ERROR 6383: Path "*string*" exists, but it is directory, not a file

ERROR 6384: Path "*string*" exists, but it is not a directory

ERROR 6387: Projection replace oid is empty or malformed

ERROR 6397: Sequences cannot be used in access policy

ERROR 6414: Subnet IP address (family *string*) "*string*" is invalid

ERROR 6421: Table name can not be empty

ERROR 6456: Unsupported authentication method - "*string*"

Vertica Documentation

Vertica Error Messages

ERROR 6473: You cannot use both parameters '*string*' and '*string*'. Client Authentication '*string*' is disabled

ERROR 6475: Can't swap partitions between these two tables: Text indices don't line up

ERROR 6492: No text index column named "*string*". Indexed text column in text index table must exist with name "*string*"

ERROR 6504: Bad expression for Access Policy

ERROR 6509: Cascade to cannot be used to make a resource pool loop

ERROR 6519: Bad expression for Access Policy - "*string*"

ERROR 6520: Can not copy access policy from table "*string*" for COLUMN "*string*": "*string*"

ERROR 6521: Can not copy access policy from table "*string*" for ROWS: "*string*"

ERROR 6545: NULL is invalid object name for analyze_external_row_count

ERROR 6546: NULL is invalid object name for drop_external_row_count)

ERROR 6566: Invalid constant hint: '*string*'

ERROR 6569: Invalid table hint identifier '*string*'

ERROR 6593: Unknown or unsupported objects

ERROR 6609: *string* process has reserved resources. Release it first

ERROR 6619: /*+syntactic_join*/ hint omitted, ignoring join hints

ERROR 6641: Argument of nth_value must be greater than zero

ERROR 6661: Can not expire password on LDAP user account

ERROR 6663: Can not modify password on LDAP user account

ERROR 6664: Can not modify profile on LDAP user account

ERROR 6666: Can not set Security Algorithm on LDAP user account

ERROR 6689: Cannot store TEMP data on HDFS storage locations

ERROR 6784: Final attempt at a database snapshot upgraded storage ids on the following nodes: *string*

ERROR 6802: Hint _oidref is empty or malformed

ERROR 6803: Ident authentication not allowed for remote connection types

ERROR 6824: Invalid explain hint identifier '*string*'

ERROR 6825: Invalid join hint identifier '*string*'

ERROR 6835: Invalid with hint identifier '*string*'

ERROR 6934: Resource Allocation for this process not found

ERROR 6944: Schema "*string*" is already set to *string* privileges

ERROR 6979: Table *string* does not contain a key constraint '*string*'

ERROR 6980: Table *string* is already set to inherit privileges

ERROR 6981: Table *string* is already set to not inherit privileges

ERROR 7042: Unknown node

ERROR 7060: User *string* could not be found in the update list

ERROR 7061: User-Defined Load function indicated that it consumed *value* bytes from the record lengths buffer, when only *value* were available

ERROR 7069: View "*string*" is already set *string* inherit privileges

ERROR 7083: /*+syntactic_join*/ hint omitted, ignoring union hints

ERROR 7102: Invalid union hint identifier '*string*'

ERROR 7104: Length of *string* type function argument cannot exceed *value* [*string*]

ERROR 7108: Parameter '*string*' cannot be NULL

Vertica Documentation

Vertica Error Messages

ERROR 7140: DataBuffer indicated that it read *value* bytes, while LengthBuffer indicated that *value* data bytes were read

ERROR 7141: DataBuffer indicated that it wrote *value* bytes, while LengthBuffer indicated that *value* data bytes were written

ERROR 7145: Invalid end-of-file state for *string*: INPUT_NEEDED

ERROR 7146: Invalid state for UDParse: OUTPUT_NEEDED

ERROR 7154: UDChunker indicated that it consumed bytes, but returned INPUT_NEEDED

ERROR 7156: Access policy cannot be created on system table "*string*"

ERROR 7159: Backup type is invalid

ERROR 7160: Cannot expand glob pattern due to error: *string*

ERROR 7162: Cannot load data due to error: *string*

ERROR 7168: Column Access Policies on flex tables may not be completely secure

ERROR 7174: Cyclic access policy detected for relation: *string*

ERROR 7179: ERROR TOLERANCE is not supported for *string* loads

ERROR 7180: ERROR TOLERANCE is not supported for ORC files

ERROR 7184: Illegal argument to create_storage_containers: NULL

ERROR 7185: Illegal argument: Please enter positive number for creating ROS containers

ERROR 7189: Occurrence parameter must be non-negative

ERROR 7194: Restore type is invalid

ERROR 7209: View *string* is already owned by *string*

ERROR 7225: Backup epoch [*value*] must be earlier than the current epoch [*value*]

ERROR 7240: Constraint '*string*' on table *string* is not a primary or unique key, nor a check constraint

ERROR 7254: ERROR TOLERANCE is not supported for Parquet files

ERROR 7269: Invalid portion: *string* [*value*]

ERROR 7271: Negative queueTimeout is not allowed

ERROR 7304: Cannot load data from [*string*]: all specified nodes are down

ERROR 7306: Cannot specify exceptions or rejected-data files on multiple nodes

ERROR 7321: Invalid action *string* specified

ERROR 7322: Invalid pool name *string* specified

ERROR 7334: Size *value* out of range

ERROR 7387: Cannot remove session's idle timeout

ERROR 7417: Error in blob creation: nChunks must be greater than 0

ERROR 7432: Invalid action URI '*string*': adapter not supported

ERROR 7434: Invalid cursor request source: *value*

ERROR 7436: Invalid number of arguments

ERROR 7451: Maxconnections cannot be greater than total number of allowable connections

ERROR 7452: Maxconnections cannot be set for a superuser

ERROR 7453: Maximum message size for notifier cannot be less than *value*

ERROR 7454: Maximum message size for notifier cannot be more than *value*

ERROR 7456: Memory size cannot be zero

ERROR 7459: Message cannot be empty

ERROR 7465: Must set mode in which connection limit is applicable

Vertica Documentation

Vertica Error Messages

ERROR 7467: Negative idlesessiontimeout is not allowed

ERROR 7468: New idlesessiontimeout *string* would exceed database wide limit of *string*

ERROR 7469: New idlesessiontimeout *string* would exceed user limit of *string*

ERROR 7472: No channel is set

ERROR 7473: No notifier is set

ERROR 7476: Notifier "*string*" does not exist

ERROR 7481: Notifier memory size (*value*) cannot be less than the maximum message size (*value*)

ERROR 7482: Notifier memory size (*value*) should be less than 2TB

ERROR 7491: Object name cannot be null

ERROR 7536: Sample Count should be between 0 and 100

ERROR 7541: Skipnode hint needs node name

ERROR 7542: Skipnode hint omitted in sub-query

ERROR 7552: Temporary data cursor request requires type information

ERROR 7564: There is no node with such name: [*string*]

ERROR 7578: UDSsource requested 0 threads on node [*string*] which is targeted for execution

ERROR 7584: Using 1 year as idlesessiontimeout

ERROR 7586: Valid value for epsilon is a positive float number smaller than 1

ERROR 7587: Valid value for outlierThreshold is a positive float number

ERROR 7615: Requested too many nodes

ERROR 7624: When an empty map of nodes is supplied, expect nNodes > 0

ERROR 7674: Idlesessiontimeout cannot be less than 15 minutes for superuser. Using 15 minutes as idlesessiontimeout

ERROR 7707: Chunk index *value* out of range 0-*value*

ERROR 7714: Error in blob creation: maxSize cannot be negative

ERROR 7715: Error in blob creation: minSize cannot be negative

ERROR 7716: Error in blob creation: minSize must be less than maxSize

ERROR 7725: Final chunk size *value* is larger than maximum chunk size *value*

ERROR 7727: Invalid notifier adapter configuration: *string*

ERROR 7741: Tried to read (*value*) past end of chunk (*value*)

ERROR 7742: Tried to read (at *value*) past end of chunk

ERROR 7743: Tried to write at offset *value* but max chunk size is *value*

ERROR 7749: Unknown notifier adapter

ERROR 7753: NULL is invalid accuracy value for analyze_histogram

ERROR 7754: NULL is invalid column name for analyze_histogram

ERROR 7770: miniBatch must be a positive integer

ERROR 7771: No resource grant exists with this id

ERROR 7799: Connection address family '*string*' is invalid; expected one of: 'ipv4','ipv6'

ERROR 7800: Connection address family '*string*' is not valid for host IP address '*string*'

ERROR 7812: Smoothing parameter alpha cannot be negative

ERROR 7868: Must specify shared storage for built-in shared file system

ERROR 7871: Projection "*string*" refers to column "*string*" referenced in the added column's default expression

ERROR 7877: Target table "string" is not anchor table for Projection "string"
ERROR 7899: Model *string* is already owned by *string*
ERROR 7910: Invalid column *string* definition for column *string*
ERROR 7911: Snapshot type is invalid
ERROR 7918: Invalid load stack: *string*
ERROR 7919: Invalid state for UDChunker::*string*(): *string*
ERROR 7924: UDChunker aligned a 0-byte chunk; chunks must be non-empty
ERROR 7933: Invalid refresh columns mode "*string*". Specify either "update" or "rebuild"
ERROR 7993: No table specified
ERROR 7995: Parameter C must be positive
ERROR 7996: Parameter epsilon must be positive
ERROR 7997: Parameter max_iterations has invalid value

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22025

This topic lists the errors associated with the SQLSTATE 22025.

SQLSTATE 22025 Description

ERRCODE_INVALID_ESCAPE_SEQUENCE

Error messages associated with this SQLState

ERROR 3656: Invalid escape sequence
ERROR 3657: Invalid escape string

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22906

This topic lists the errors associated with the SQLSTATE 22906.

SQLSTATE 22906 Description

ERRCODE_NONSTANDARD_USE_OF_ESCAPE_CHARACTER

Error messages associated with this SQLState

ERROR 4166: Nonstandard use of \ ' in a string literal at or near "string"

ERROR 4167: Nonstandard use of \\ in a string literal at or near "string"

ERROR 4168: Nonstandard use of escape in a string literal at or near "string"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V02

This topic lists the errors associated with the SQLSTATE 22V02.

SQLSTATE 22V02 Description

ERRCODE_INVALID_TEXT_REPRESENTATION

Error messages associated with this SQLState

ERROR 2825: Could not convert "string"string to a boolean

ERROR 2826: Could not convert "string"string to a float8

ERROR 2827: Could not convert "string"string to an int8

ERROR 3677: Invalid input for *string*: "*string*"
ERROR 3680: Invalid input syntax for boolean: "*string*"
ERROR 3681: Invalid input syntax for integer: "*string*"
ERROR 3682: Invalid input syntax for numeric: "*value*"
ERROR 3711: Invalid number: "*string*"
ERROR 3712: Invalid numeric format *string*
ERROR 3714: Invalid numeric value: "*string*"
ERROR 3751: Invalid Session ID format
ERROR 3757: Invalid syntax for float: "*string*"
ERROR 3758: Invalid syntax for numeric: "*string*"
ERROR 3894: Malformed record literal: "*string*"
ERROR 4169: Not a number: "*string*"
ERROR 4198: Number exceeds format: "*string*"
ERROR 5930: Invalid numeric format *string*. Expected precision is *value* and scale is *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V03

This topic lists the errors associated with the SQLSTATE 22V03.

SQLSTATE 22V03 Description

ERRCODE_INVALID_BINARY_REPRESENTATION

Error messages associated with this SQLState

ERROR 2829: Could not convert integer *valuestring* to a boolean
ERROR 3536: Incorrect binary data format in bind parameter *value*
ERROR 3623: Invalid binary input syntax: '*value*'
ERROR 3624: Invalid bitstring "*string*"
ERROR 3671: Invalid hex string "*string*"
ERROR 3678: Invalid input syntax for *string*
ERROR 3716: Invalid octal string format "*string*"
ERROR 3717: Invalid octal string format (octal string length

ERROR 5416: Value too long for type *string(value)*

ERROR 5936: Invalid string format "*string*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V04

This topic lists the errors associated with the SQLSTATE 22V04.

SQLSTATE 22V04 Description

ERRCODE_BAD_COPY_FILE_FORMAT

Error messages associated with this SQLState

ERROR 2031: *string* Header size (*value*) is corrupted

ERROR 2032: *string* Header size (*value*) is too small

ERROR 2035: *string* Input record *value* has been rejected (*string*)

ERROR 2053: *string* Row size (*value*) is corrupted

ERROR 2054: *string* Unexpected EOF while reading header. Expected *value* but read *value*

ERROR 2738: COPY file signature not recognized

ERROR 2767: COPY: Wrong Header size *value*. Expected *value*

ERROR 3562: Input has extra trailing bytes

ERROR 3640: Invalid COPY file header (unsupported Version Number)

ERROR 4206: Number of fields is *value*, expected *value*

ERROR 4627: Row delimiter not found; corrupt file input (read *value* bytes from input)

ERROR 5495: Wrong size *value* for bool column *value* (*string*)

ERROR 5496: Wrong size *value* for date column *value* (*string*)

ERROR 5497: Wrong size *value* for float column *value* (*string*)

ERROR 5498: Wrong size *value* for integer column *value* (*string*)

ERROR 5499: Wrong size *value* for Interval column *value* (*string*)

ERROR 5500: Wrong size *value* for Numeric column *value* (*string*)

ERROR 5501: Wrong size *value* for Time column *value* (*string*)

ERROR 5502: Wrong size *value* for Timestamp column *value* (*string*)

ERROR 5503: Wrong size *value* for TimestampTz column *value* (*string*)

ERROR 5504: Wrong size *value* for TimeTz column *value* (*string*)

ERROR 6363: Not able to skip *value* rows for source [*string*]

ERROR 7293: *string* [*value*] records have been rejected

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V0B

This topic lists the errors associated with the SQLSTATE 22V0B.

SQLSTATE 22V0B Description

ERRCODE_ESCAPE_CHARACTER_ON_NOESCAPE

Error messages associated with this SQLState

ERROR 2746: COPY NO ESCAPE cannot also contain an ESCAPE clause

ERROR 2747: COPY NO ESCAPE for column *string* cannot also contain an ESCAPE clause

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V21

This topic lists the errors associated with the SQLSTATE 22V21.

SQLSTATE 22V21 Description

ERRCODE_INVALID_EPOCH

Error messages associated with this SQLState

ERROR 2144: AHM can't advance past the cluster last full backup epoch. (Last Backup Epoch: *value*)

ERROR 2145: AHM can't advance past the cluster last backup time. (Last Backup time: *string*)

ERROR 2146: AHM can't advance past the cluster last good epoch (LGE) time (Cluster LGE time: *string*)

ERROR 2147: AHM can't advance past the cluster last good epoch (LGE). (Cluster LGE: *value*)

ERROR 2148: AHM can't advance past the latest epoch time (Latest epoch time: *string*)

ERROR 2153: AHM must be less than the current epoch (Current Epoch: *value*)

ERROR 2318: Can't run historical queries at epochs prior to the Ancient History Mark

ERROR 3000: DDL interfered with this statement

ERROR 3184: Epoch specified is not in historical epoch range

ERROR 3559: Input epoch must be greater than or equal to the earliest epoch (earliest epoch: *value*)

ERROR 3560: Input epoch must be greater than the current AHM (Current AHM: *value*)

ERROR 3561: Input epoch must be less than or equal to the AHM epoch (AHM epoch: *value*)

ERROR 3567: Input time can't be rounded down to an epoch higher than the current AHM epoch (Current AHM epoch: *value*, Current AHM time: *string*)

ERROR 3568: Input time must be greater than or equal to the earliest epoch time (Earliest epoch time: *string*)

ERROR 3569: Input time must be greater than the current AHM time (Current AHM time: *string*)

ERROR 3570: Input time must be less than or equal to the AHM epoch time (AHM epoch time: *string*)

ERROR 3654: Invalid epoch

ERROR 3844: Last good epoch not set

ERROR 3926: MergeOut start epoch (*=value*) greater than end epoch (*=value*)

ERROR 4940: The current AHM is already *value*

ERROR 5013: Time specified is not in historical epoch range

ERROR 7639: AHM can't advance due to lack of full backup

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V23

This topic lists the errors associated with the SQLSTATE 22V23.

SQLSTATE 22V23 Description

ERRCODE_RAISE_EXCEPTION

Error messages associated with this SQLState

ERROR 5783: Client error: *string* (in function *string()* at *string:value*)

ERROR 7084: ABORTRECOVERY is only allowed in UNSAFE mode

ERROR 7136: USER GENERATED ERROR

ERROR 7137: USER GENERATED ERROR: *value*

ERROR 7804: LGE is lagging behind for *value* seconds, it exceeds *value* seconds threshold, LGE: *value*, current epoch: *value*

ERROR 7813: Snapshot failed because some projections unable to move out data to ROS. Minimum checkpoint epoch of projections in snapshot: *value*, current epoch: *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 22V24

This topic lists the errors associated with the SQLSTATE 22V24.

SQLSTATE 22V24 Description

ERRCODE_COPY_PARSE_ERROR

Error messages associated with this SQLState

ERROR 6703: Corrupt orc source!

ERROR 6723: Data type *string* is not supported for the ORC file format

ERROR 6726: Datatype mismatch: column *value* in the orc source [*string*] has type *string*, expected *string*

ERROR 6777: Failed to read orc source [*string*]

ERROR 6778: Failed to read orc source [*string*]: *string*

ERROR 7087: Attempt to load *value* columns from an orc source [*string*] that has *value* columns

ERROR 7095: Failed to read orc source [*string*]: complex types are not supported

ERROR 7127: The table has *value* columns, but the orc source [*string*] has *value* columns

ERROR 7157: Attempt to load *value* columns from an orc source [*string*] that has *value* columns and *value* partition columns

ERROR 7202: The table has *value* columns, but the orc source [*string*] has *value* columns and *value* partition columns

ERROR 7223: Attempt to load *value* columns from a parquet source [*string*] that has *value* columns

ERROR 7224: Attempt to load *value* columns from a parquet source [*string*] that has *value* columns and *value* partition columns

ERROR 7241: Corrupt parquet source!

ERROR 7246: Data type *string* is not supported for the PARQUET file format

ERROR 7247: Datatype mismatch: column *value* in the parquet source [*string*] has type *string*, expected *string*

ERROR 7257: Failed to read parquet source [*string*]

ERROR 7258: Failed to read parquet source [*string*]: *string*

ERROR 7259: Failed to read parquet source [*string*]: complex types are not supported

ERROR 7286: The table has *value* columns, but the parquet source [*string*] has *value* columns

ERROR 7287: The table has *value* columns, but the parquet source [*string*] has *value* columns and *value* partition columns

ERROR 7786: Failed to read parquet source [*string*]: some of the *value* selected columns do not match with *value* columns in file

ERROR 7985: Decimal type with precision *value* (> 38) is not supported

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 23502

This topic lists the errors associated with the SQLSTATE 23502.

SQLSTATE 23502 Description

ERRCODE_NOT_NULL_VIOLATION

Error messages associated with this SQLState

ERROR 2416: Cannot drop NOT NULL constraint on column "*string*" when it is referenced in PARTITION BY expression

ERROR 2417: Cannot drop NOT NULL constraint on column "*string*" when it is referenced in primary key constraint

ERROR 2623: Column "*string*" definition changed to NOT NULL

ERROR 4182: NOT NULL constraint on column "*string*" already exists

ERROR 4183: NOT NULL constraint on column "*string*" does not exist

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 23503

This topic lists the errors associated with the SQLSTATE 23503.

SQLSTATE 23503 Description

ERRCODE_FOREIGN_KEY_VIOLATION

Error messages associated with this SQLState

ERROR 4165: Nonexistent foreign key value detected in FK-PK join [*string*]; value [*string*]

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 23505

This topic lists the errors associated with the SQLSTATE 23505.

SQLSTATE 23505 Description

ERRCODE_UNIQUE_VIOLATION

Error messages associated with this SQLState

ERROR 3147: Duplicate MERGE key detected in join [*string*]; value [*string*]

ERROR 3149: Duplicate primary/unique key detected in join [*string*]; value [*string*]

ERROR 4840: Subquery used as an expression returned more than one row

ERROR 6744: Duplicate key values in table '*string*': '*string*' -- violates constraint '*string*'

ERROR 6745: Duplicate key values: '*string*' -- violates constraint '*string*'

ERROR 7695: Null value in primary key column: '*string*' -- violates constraint '*string*'

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 23514

This topic lists the errors associated with the SQLSTATE 23514.

SQLSTATE 23514 Description

ERRCODE_CHECK_VIOLATION

Error messages associated with this SQLState

ERROR 7230: Check constraint '*value*' value violation: '*value*'

ERROR 7231: Check constraint '*string*' string violation in table '*string*': '*string*'

ERROR 7232: Check constraint '*string*' string violation: '*string*'

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 25V01

This topic lists the errors associated with the SQLSTATE 25V01.

SQLSTATE 25V01 Description

ERRCODE_NO_ACTIVE_SQL_TRANSACTION

Error messages associated with this SQLState

ERROR 2342: Cannot advance epoch without a transaction

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 28000

This topic lists the errors associated with the SQLSTATE 28000.

SQLSTATE 28000 Description

ERRCODE_INVALID_AUTHORIZATION_SPECIFICATION

Error messages associated with this SQLState

ERROR 2701: Conflicting, redundant or unsupported option: *string*

ERROR 2702: Conflicting, redundant or unsupported option: *groupElts*

ERROR 2959: Current user cannot be dropped

ERROR 4293: Option "*string*" not recognized

ERROR 4722: Session user cannot be dropped

ERROR 4846: Superuser cannot be dropped
ERROR 5387: User does not exist
ERROR 6441: Trying to change password, but user "string" does not exist
ERROR 6442: Trying to find authentication mehtod, but user "string" does not exist
ERROR 6443: Trying to update login history, but user "string" does not exist
ERROR 6494: SecurityAlgorithm valid options are NONE, MD5 and SHA512
ERROR 6671: Cannot change password on LDAP user "string"
ERROR 6841: LDAP role "string" cannot be dropped unless it is orphaned (sourced from a different LDAP than currently configured)
ERROR 6842: LDAP user "string" cannot be renamed
ERROR 6843: LDAP user "string" has not database password
ERROR 6844: LDAP user cannot be dropped unless it is orphaned
ERROR 7667: SecurityAlgorithm valid options are NONE and SHA512

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 2BV01

This topic lists the errors associated with the SQLSTATE 2BV01.

SQLSTATE 2BV01 Description

ERRCODE_DEPENDENT_OBJECTS_STILL_EXIST

Error messages associated with this SQLState

ERROR 3052: Dependent privileges exist
ERROR 3128: DROP failed due to dependencies
ERROR 3130: DROP PROFILE failed due to dependencies
ERROR 3131: DROP ROLE failed due to dependencies
ERROR 6240: DROP AUTHENTICATION failed due to dependencies
ERROR 7303: Cannot drop table "string" because other objects depend on it
ERROR 7345: *string* failed due to dependencies
ERROR 7945: Column *string* is dropped which will break some dependent access policies on the table.
Certain queries may fail

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 38004

This topic lists the errors associated with the SQLSTATE 38004.

SQLSTATE 38004 Description

ERRCODE_E_R_E_READING_SQL_DATA_NOT_PERMITTED

Error messages associated with this SQLState

ERROR 7339: User may not have read privilege to the external table storage location: [*string*]

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 40V01

This topic lists the errors associated with the SQLSTATE 40V01.

SQLSTATE 40V01 Description

ERRCODE_T_R_DEADLOCK_DETECTED

Error messages associated with this SQLState

ERROR 3010: Deadlock: *string* - *string*

ERROR 3011: Deadlock: [Txn value] *string* - *string* error *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42501

This topic lists the errors associated with the SQLSTATE 42501.

SQLSTATE 42501 Description

ERRCODE_INSUFFICIENT_PRIVILEGE

Error messages associated with this SQLState

ERROR 2065: *string*: Invalid table/projection/column *string*

ERROR 2198: analyze_statistics: Requires modify permissions for table/projection/column *string*

ERROR 2347: Cannot alter predefined role "*string*"

ERROR 2348: Cannot alter superuser *string*'s default roles

ERROR 2349: Cannot alter superuser roles

ERROR 2389: Cannot create system built-in tuning rule

ERROR 2419: Cannot drop system built-in tuning rule

ERROR 2460: Cannot move user *string* to general pool, they lack privileges

ERROR 2481: Cannot remove memoryCap

ERROR 2482: Cannot remove runTimeCap

ERROR 2484: Cannot remove tempSpaceCap

ERROR 2515: Cannot set resource pool: user *string* lacks privileges on resource pool *string*

ERROR 2812: Could not add location [*string*]: Permission denied

ERROR 2935: Couldn't nice(*value*) thread: *value*

ERROR 2953: Current password must be supplied to set new password

ERROR 2958: Current user can't change runtime priority of another user's task

ERROR 2960: Current user doesn't have the privilege to change the task runtime priority to be higher than its resource pool

ERROR 3577: Insufficient permissions on projection "*string*"

ERROR 3578: Insufficient permissions on schema "*string*"

ERROR 3579: Insufficient permissions on table "*string*"

ERROR 3580: Insufficient privilege: USAGE on SCHEMA '*string*' not granted for current user

ERROR 3581: Insufficient privileges for projection *string*

Vertica Documentation

Vertica Error Messages

ERROR 3582: Insufficient privileges for table *string*

ERROR 3583: Insufficient privileges on *string*

ERROR 3584: Insufficient privileges on *string*, modify privileges (INSERT|UPDATE|DELETE|TRUNCATE) needed

ERROR 3722: Invalid passphrase: *string*

ERROR 3919: memoryCap of *value* KB would exceed user limit of *value* KB

ERROR 3989: Must be owner of *string string*

ERROR 3990: Must be owner of *string [string]*

ERROR 3991: Must be superuser to alter database

ERROR 3992: Must be superuser to alter profile

ERROR 3993: Must be superuser to alter tuning rule

ERROR 3994: Must be superuser to alter user default roles

ERROR 3998: Must be superuser to clear Query/EE profiles

ERROR 4000: Must be superuser to create interface

ERROR 4001: Must be superuser to create library

ERROR 4002: Must be superuser to create profile

ERROR 4003: Must be superuser to create subnet

ERROR 4004: Must be superuser to create tuning rule

ERROR 4005: Must be superuser to create users

ERROR 4006: Must be superuser to drop an interface

ERROR 4007: Must be superuser to drop library

ERROR 4008: Must be superuser to drop profile

ERROR 4009: Must be superuser to drop resource pool

ERROR 4010: Must be superuser to drop role

ERROR 4011: Must be superuser to drop subnet

ERROR 4012: Must be superuser to drop tuning rule

ERROR 4013: Must be superuser to drop users

ERROR 4014: Must be superuser to modify resource pools

ERROR 4015: Must be superuser to rename interface

ERROR 4016: Must be superuser to rename profile

ERROR 4017: Must be superuser to rename role

ERROR 4018: Must be superuser to rename subnet

ERROR 4019: Must be superuser to run *string*

ERROR 4020: Must be superuser to run *analyze_workloadstring()*

ERROR 4059: New runTimeCap *value* ms would exceed user limit of *value* ms

ERROR 4061: New tempSpaceCap *value* KB would exceed user limit of *value* KB

ERROR 4178: Not enough privileges for projection *string*

ERROR 4179: Not enough privileges for table *string*

ERROR 4244: Only database superuser can drop procedures

ERROR 4260: Only superuser can check privileges on other users

ERROR 4261: Only superuser can create roles

ERROR 4269: Only the database super user can create procedures

ERROR 4366: Permission denied

Vertica Documentation

Vertica Error Messages

ERROR 4367: Permission denied for *string string*

ERROR 4368: Permission denied for *string [string]*

ERROR 4369: Permission denied to create temporary tables

ERROR 4370: Permission denied: "*string*" is a system catalog

ERROR 4546: RecvFiles on *string*: Can't write to file [*string*]

ERROR 4741: setThreadCPUNiceValue: couldn't nice(*value*) thread: *value*

ERROR 4742: setThreadIONiceValue: couldn't ionice(*value*) thread: *value*

ERROR 5149: Unable to set role "*string*"

ERROR 5389: User has insufficient privileges on schema *string*

ERROR 5488: Workspace schema *string* does not exist

ERROR 5517: Your Vertica license is invalid or has expired

ERROR 5618: Must be superuser to alter fault group

ERROR 5619: Must be superuser to create fault group

ERROR 5620: Must be superuser to drop fault group

ERROR 5635: Path to file [*string*] contains a symbolic link

ERROR 5716: Must have create permissions in schema *string* to drop type

ERROR 5818: Deployment script will not be generated since the user does not have appropriate permissions to write to [*string*]

ERROR 5820: Design script will not be generated since the user does not have appropriate permissions to write to [*string*]

ERROR 5956: Must be superuser to ALTER NODE

ERROR 5957: Must be superuser to create filesystem

ERROR 5958: Must be superuser to create location

ERROR 5959: Must be superuser to CREATE NODEs

ERROR 5961: Must be superuser to supply 'user_name' argument to HAS_ROLE() function
HINT: Non-superusers run HAS_ROLE('role_name')

ERROR 5975: Not enough privileges for *string*

ERROR 6125: Access Policies are not enabled

ERROR 6343: Must be superuser to *string* authentication

ERROR 6344: Must be superuser to alter access policy

ERROR 6345: Must be superuser to alter authentication

ERROR 6346: Must be superuser to copy access policy

ERROR 6347: Must be superuser to create access policy

ERROR 6348: Must be superuser to create authentication

ERROR 6349: Must be superuser to drop access policy

ERROR 6350: Must be superuser to drop authentication

ERROR 6351: Must be superuser to rename authentication

ERROR 6514: Must be superuser to run *value* <name not available>

ERROR 6540: *string*: Invalid external table *string*

ERROR 6871: Must be superuser to rename user

ERROR 6872: Must be superuser to run move_statement_to_resource_pool

ERROR 6896: Parameter *string* can be cleared only via a UDX

ERROR 6897: Parameter *string* can be set only via a UDX

ERROR 6902: Permission denied on [*string.string*]
ERROR 6903: Permission denied, must be superuser to create a directed query
ERROR 6904: Permission denied, must be superuser to drop a directed query
ERROR 6905: Permission denied, must be superuser to save a query
ERROR 6906: Permission denied, must be superuser to update directed query status
ERROR 7376: Cannot close other user's sessions
ERROR 7463: Must be superuser to create notifier
ERROR 7464: Must be superuser to drop notifier
ERROR 7998: Permission denied for model *string*
ERROR 7999: Permission denied for schema *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42601

This topic lists the errors associated with the SQLSTATE 42601.

SQLSTATE 42601 Description

ERRCODE_SYNTAX_ERROR

Error messages associated with this SQLState

ERROR 2030: *string* has been deprecated as *string string* Vertica option
ERROR 2069: '*string*' is not a table name in the current search_path
ERROR 2085: A column cannot occur in an equality predicate and an interpolation predicate
ERROR 2086: A column definition list is only allowed for functions that return "record"
ERROR 2087: A column definition list is required for functions returning "record"
ERROR 2093: A join can have only one set of interpolated predicates
ERROR 2100: A query with Time Series Aggregate Function *string* must have a timeseries clause
ERROR 2156: All columns are evaluated by expressions. At least one column should be read from input
ERROR 2157: All columns in select list must be columns used by projection
ERROR 2164: Alter Column Type driver: Unrecognized command type
ERROR 2180: Analytic function *string* must have an OVER clause
ERROR 2187: Analytic functions cannot be nested

Vertica Documentation

Vertica Error Messages

ERROR 2191: ANALYZE CONSTRAINT is not supported

ERROR 2203: Anchor table not found

ERROR 2214: Argument *value* has invalid type *value* in ANALYZE_WORKLOAD

ERROR 2215: Argument *value* in ANALYZE_WORKLOAD must be constant

ERROR 2223: Argument in ANALYZE_CONSTRAINTS must be constant

ERROR 2230: Arguments of row IN must all be row expressions

ERROR 2238: At least two arguments are required

ERROR 2239: At most one path number can be entered

ERROR 2346: Cannot alter a sequence with START

ERROR 2374: Cannot compare row expressions of zero length

ERROR 2381: Cannot create a sequence with RESTART

ERROR 2444: Cannot insert into or update IDENTITY/AUTO_INCREMENT column "*string*"

ERROR 2445: Cannot insert into system column "*string*"

ERROR 2446: Cannot insert multiple commands into a prepared statement

ERROR 2521: Cannot specify anything other than user defined transforms *string* in the *string* list

ERROR 2525: Cannot specify more than one user-defined transform function in the SELECT list

ERROR 2526: Cannot specify more than one window clause with a user defined transform

ERROR 2534: Cannot use "PR" with "S"/"PL"/"MI"/"SG"

ERROR 2535: Cannot use "S" with "MI"

ERROR 2536: Cannot use "S" with "PL"

ERROR 2537: Cannot use "S" with "PL"/"MI"/"SG"/"PR"

ERROR 2538: Cannot use "S" with "SG"

ERROR 2539: Cannot use "V" with a decimal point

ERROR 2545: Cannot use aggregate function in VALUES

ERROR 2627: Column "*string*" in ENCODED BY clause is not found in the table

ERROR 2641: Column "*string.string*" must appear in the PARTITION BY list of Timeseries clause or be used in a Time Series Aggregate Function

ERROR 2642: Column *string* cannot be evaluated

ERROR 2645: Column *string* has other computed columns in its expression

ERROR 2647: Column *string* in ORDER BY list is not found in TABLE

ERROR 2659: Column alias list for "*string*" has too many entries

ERROR 2669: COLUMN OPTION is not supported

ERROR 2670: Column options are not supported

ERROR 2696: Conflicting INTERVAL subtypes

ERROR 2697: Conflicting NULL/NOT NULL declarations for column "*string*" of table "*string*"

ERROR 2715: Constraint declared INITIALLY DEFERRED must be DEFERRABLE

ERROR 2754: COPY requires a data source; either a FROM clause or a WITH SOURCE for a user-defined source

ERROR 2764: COPY: Expression for column *string* cannot be coerced

ERROR 2946: CREATE TABLE AS specifies too many column names

ERROR 2947: CREATE VIEW specifies more column names than columns

ERROR 2986: Database name is required (too few dotted names): *string*

ERROR 3023: Default values specified for IDENTITY/AUTO_INCREMENT column "*string*" of table "*string*"

Vertica Documentation

Vertica Error Messages

ERROR 3125: Drop Column driver: Unrecognized command type

ERROR 3142: Duplicate column "*string*" in create table statement

ERROR 3143: Duplicate column *string* in constraint

ERROR 3146: Duplicate columns in select list of projection not allowed

ERROR 3151: Duplicate tables in projection not allowed

ERROR 3155: Duplicated parameters *string* not allowed

ERROR 3158: Each *string* query must have the same number of columns

ERROR 3164: Empty column name is invalid

ERROR 3165: Empty constraint name is invalid

ERROR 3171: ENCODED BY is not supported in CREATE PROJECTION statement when column renaming list is defined

ERROR 3172: ENCODED BY is not supported in CREATE PROJECTION statement with column definition list

ERROR 3173: ENCODED BY is not supported in CREATE TABLE AS SELECT statement when column list is defined

ERROR 3176: End epoch (*value*) number out of range

ERROR 3177: End epoch (*value*) precedes start epoch (*value*)

ERROR 3183: Epoch number out of range

ERROR 3185: Epoch time out of range

ERROR 3261: Error setting *string* in *string*: Unknown Property

ERROR 3262: Error setting basic directives: '*string*'

ERROR 3263: Error setting designer directives: '*string*'

ERROR 3264: Error setting optimizer directives: '*string*'

ERROR 3344: EXPORT ... SELECT may not specify INTO

ERROR 3348: Expression "(*<string>* - *<string>*) *<interval qualifier>*" is not supported

ERROR 3349: Expression for column *string* cannot be coerced

ERROR 3458: Function *string* is not allowed in Time Series queries

ERROR 3461: Function *string* requires at least one argument

ERROR 3487: Group by is not allowed in a projection

ERROR 3500: HAVING / GROUP BY not allowed with Time Series query

ERROR 3511: IGNORE NULLS can only be used with FIRST_VALUE or LAST_VALUE or NTH_VALUE

ERROR 3517: Improper %TYPE reference (too few dotted names): *string*

ERROR 3518: Improper %TYPE reference (too many dotted names): *string*

ERROR 3519: Improper qualified column name: *string*

ERROR 3520: Improper qualified name (too many dot): *string*

ERROR 3521: Improper qualified name (too many dots): *string*

ERROR 3522: Improper qualified name (too many dotted names): *string*

ERROR 3523: Improper relation name (too many dotted names): *string*

ERROR 3538: Incorrect parameter type provided: *string* is supposed to be of type *string*

ERROR 3548: Indirection is not allowed in a target column

ERROR 3549: Indirection is not allowed in the name of a FILLER column

ERROR 3571: INSERT ... SELECT may not specify INTO

ERROR 3572: INSERT has more expressions than target columns

ERROR 3573: INSERT has more target columns than expressions

Vertica Documentation

Vertica Error Messages

ERROR 3599: Interpolated predicates are allowed only in ON CLAUSE of ANSI Join syntax

ERROR 3602: Interpolated predicates should refer to columns from both relations of the join

ERROR 3615: INTO is only allowed on first SELECT of UNION/INTERSECT/EXCEPT

ERROR 3619: Invalid argument type *value* in ANALYZE_CONSTRAINTS

ERROR 3672: Invalid hexadecimal number at or near "*string*"

ERROR 3706: Invalid node name in hint

ERROR 3709: Invalid number at or near "*string*"

ERROR 3738: Invalid projection name in hint: *string*

ERROR 3775: Invalid Unicode escape character '*character*'

ERROR 3776: Invalid Unicode hex number "*string*"

ERROR 3812: Join condition in merge query must include at least one table attribute

ERROR 3841: Label can accept only one argument

ERROR 3865: LIMIT #,# syntax is not supported

ERROR 3944: Misplaced DEFERRABLE clause

ERROR 3945: Misplaced INITIALLY DEFERRED clause

ERROR 3946: Misplaced INITIALLY IMMEDIATE clause

ERROR 3947: Misplaced NOT DEFERRABLE clause

ERROR 3949: Missing argument

ERROR 3958: Missing savepoint name

ERROR 3959: Missing the path number

ERROR 3976: Multiple assignments to same column "*string*"

ERROR 3978: Multiple decimal points

ERROR 3979: Multiple default values specified for column "*string*" of table "*string*"

ERROR 3980: Multiple DEFERRABLE/NOT DEFERRABLE clauses not allowed

ERROR 3981: Multiple FOR UPDATE clauses are not allowed

ERROR 3982: Multiple INITIALLY IMMEDIATE/DEFERRED clauses not allowed

ERROR 3984: Multiple LIMIT clauses are not allowed

ERROR 3985: Multiple OFFSET clauses are not allowed

ERROR 3986: Multiple ORDER BY clauses are not allowed

ERROR 4023: Must specify memorySize parameter

ERROR 4024: Must specify one new name for each schema

ERROR 4025: Must specify one new name for each table

ERROR 4026: Must specify one new name for each view

ERROR 4062: NEW used in query that is not in a rule

ERROR 4066: No actions specified

ERROR 4070: No columns specified in select list

ERROR 4072: No constraints defined

ERROR 4105: No second argument needed when analyzing all constraints

ERROR 4136: Node "*string*" does not exist

ERROR 4161: Non-integer constant in *string*

ERROR 4164: Nonexistent columns: '*string*'

ERROR 4203: Number of columns defined in CREATE TABLE statement is less than in SELECT query output

Vertica Documentation

Vertica Error Messages

ERROR 4204: Number of columns defined in CREATE TABLE statement is more than in SELECT query output

ERROR 4205: Number of columns in the PROJECTION statement must be the same as the number of columns in the SELECT statement

ERROR 4225: OLD used in query that is not in a rule

ERROR 4227: ON COMMIT clause may only be specified for TEMPORARY tables

ERROR 4237: Only a single "S" is allowed

ERROR 4239: Only ASC is allowed in ORDER BY list of auto projection for CREATE TABLE

ERROR 4240: Only columns are allowed in ORDER BY list of auto projection for CREATE TABLE

ERROR 4241: Only columns are allowed in SELECT list of projection

ERROR 4247: Only inner joins are allowed in a projection defining query

ERROR 4268: Only tables are allowed in FROM clause of projection

ERROR 4291: Operator too long at or near "*string*"

ERROR 4294: Option *string* conflicts with prior options

ERROR 4296: Options not set

ERROR 4297: ORDER BY column in timeseries OVER clause must be Timestamp type

ERROR 4325: Parameters can only contain constants or constant expressions

ERROR 4327: Parsing error "*string*" at or near "*string*"

ERROR 4348: Path Number must be in [0, *value*]

ERROR 4350: Pattern "0" must come before "PR"

ERROR 4351: Pattern "9" must come before "PR"

ERROR 4383: plannedConcurrency must be greater than 0

ERROR 4487: Projections can only be sorted in ascending order

ERROR 4629: Row expressions being compared must have the same number of entries

ERROR 4669: SELECT * with no tables specified is not valid

ERROR 4670: SELECT DISTINCT ON is not standard SQL, use just SELECT DISTINCT

ERROR 4706: Sequence functions accept constant strings arguments only

ERROR 4707: Sequence Manipulation functions are allowed in OUTER SELECT LIST only and cannot be in SELECT LIST of a WITH clause

ERROR 4732: Set Operators are not allowed in a projection

ERROR 4761: Sort key *string* should be in the target list

ERROR 4814: Subqueries in a MERGE statement are not allowed

ERROR 4815: Subqueries in MERGE statement are not allowed

ERROR 4828: Subquery has too few columns

ERROR 4829: Subquery has too many columns

ERROR 4831: Subquery in FROM may not have SELECT INTO

ERROR 4833: Subquery in FROM must have an alias

ERROR 4835: Subquery must return a column

ERROR 4836: Subquery must return only one column

ERROR 4837: Subquery not allowed in a projection

ERROR 4838: Subquery not allowed in SELECT list and/or ORDER BY clause for Time Series queries

ERROR 4855: Syntactic Optimizer requires joins written using ANSI JOIN syntax

ERROR 4856: Syntax error at or near "*string*"

Vertica Documentation

Vertica Error Messages

ERROR 4947: The foreign key in this constraint has already been defined as a foreign key for relation *"string"*

ERROR 4955: The number of target columns (*value*) does not match the number of columns (*value*) in the EXPORT statement

ERROR 4956: The number of target columns (*value*) is less than the number of columns (*value*) in the EXPORT statement

ERROR 5007: Time Series Aggregate Functions cannot be nested

ERROR 5008: Time Series queries cannot refer to column of outer query

ERROR 5009: Time Series queries cannot refer to column of outer query: *"string.string"*

ERROR 5011: Time slice length must be a positive integer constant

ERROR 5012: Time slice length must be an interval constant

ERROR 5161: Unequal number of entries in row expression

ERROR 5162: Unequal number of entries in row expressions

ERROR 5272: Unsupported From clause expression

ERROR 5285: Unsupported SET option

ERROR 5286: Unsupported SET option *string*

ERROR 5287: Unsupported SHOW option *string*

ERROR 5290: Unsupported transaction option *string*

ERROR 5305: Unterminated /* comment at or near *"string"*

ERROR 5306: Unterminated bit string literal at or near *"string"*

ERROR 5307: Unterminated dollar-quoted string at or near *"string"*

ERROR 5308: Unterminated hexadecimal string literal at or near *"string"*

ERROR 5310: Unterminated quoted identifier at or near *"string"*

ERROR 5311: Unterminated quoted string at or near *"string"*

ERROR 5323: Usage: `clear_profiling(string , string)`

ERROR 5324: Usage: `disable_profiling(string)`

ERROR 5325: Usage: `enable_profiling(string)`

ERROR 5326: Use *"string(*)"* to call this aggregate function

ERROR 5383: User Defined Transform Functions are allowed only in a SELECT list

ERROR 5386: User defined transform will return *value* columns, whereas *value* aliases provided

ERROR 5401: User-defined transform function *string* must have an OVER clause

ERROR 5413: Value must be either "units" or "plain"

ERROR 5415: Value must be either ON or OFF

ERROR 5452: Virtual tables are not allowed in FROM clause of projection

ERROR 5492: Wrong number of parameters for prepared statement *"string"*

ERROR 5493: Wrong number of parameters on left side of OVERLAPS expression

ERROR 5494: Wrong number of parameters on right side of OVERLAPS expression

ERROR 5505: You can specify a node name only once in a create projection statement, node *string* appears more than once

ERROR 5518: Zero-length delimited identifier at or near *"string"*

ERROR 5524: A projection can have only one basename

ERROR 5525: A projection can have only one createtype

ERROR 5566: Dimension tables may not have data that shorter lived than the fact table

Vertica Documentation

Vertica Error Messages

ERROR 5577: Expression for user-defined type column *string* cannot be coerced

ERROR 5600: Invalid predicate in projection-select. Only PK=FK equijoins are allowed

ERROR 5617: Multiple WITH clauses not allowed

ERROR 5629: Not a Star or Snow-Flake Query

ERROR 5630: Nullable FKs are not allowed in projection definition

ERROR 5651: Recursive With is not supported

ERROR 5664: Subqueries not allowed in projection definition

ERROR 5665: Subquery in MERGE is not supported

ERROR 5670: The number of alias columns must be the same as the number of selected columns

ERROR 5691: User-defined function *string* is not a supported scalar function

ERROR 5696: WITH query name "*string*" specified more than once

ERROR 5711: Invalid function arguments

ERROR 5714: Missing the random seed

ERROR 5730: The second argument, sampling method, should be always be 1 -- naive sampling(biased)

ERROR 5733: The third argument must be large than 0

ERROR 5734: Three arguments at most: sampling seed, sampling method (optional, default 1), sampling size (optional,default 10)

ERROR 5916: If specified, maximum error percentage must be a numeric constant

ERROR 5926: Internal error parsing function *string*

ERROR 5929: Invalid maximum error percentage specified

ERROR 6034: Syntax Error: '*string*' is a built in type

ERROR 6048: The minimum value that may be specified for maximum error percentage is 0.88

ERROR 6061: Too many arguments to *string*

ERROR 6119: 'deleted' hint takes no arguments

ERROR 6120: 'latestdata' hint arguments must be strings

ERROR 6122: A projection can replace only one original

ERROR 6124: A single argument must be supplied to *string*

ERROR 6136: Aggregate projection without group-by columns is not supported

ERROR 6137: Aggregate projections must have at least one aggregate (SUM, COUNT, MIN, or MAX)

ERROR 6140: All sort keys should be in the target list of the projection

ERROR 6146: At least two columns must be specified

ERROR 6196: Columns in ORDER-BY must be defined in the SELECT statement

ERROR 6197: Columns/Expressions in GROUP BY/PARTITION BY must be first and in the same order with columns in SELECT

ERROR 6198: ORDER BY columns/expressions in the OVER() clause must be the first SELECT columns/expressions not specified by PARTITION BY clause, and must be specified in SELECT list order

ERROR 6199: Columns/Expressions in the PARTITION BY clause may not be repeated in the ORDER BY clause

ERROR 6232: DISTINCT Aggregates are not allowed in projection

ERROR 6293: Interpolated join predicates cannot have expressions or functions over join columns

ERROR 6294: Invalid Argument

ERROR 6321: Limit in Top-K query must be a positive number

ERROR 6324: Limit/Offset is only allowed in topk projections

ERROR 6340: Multiple ORDER BY clauses in partition TopK/Limit query are not allowed

Vertica Documentation

Vertica Error Messages

ERROR 6341: Multiple PARTITION clauses in TopK/Limit query are not allowed

ERROR 6366: Only one table or projection is allowed in FROM clause of aggregate projection

ERROR 6367: Only one table or projection is allowed in FROM clause of top-k projection

ERROR 6369: Only SUM, MIN, MAX and COUNT are allowed in aggregate projections

ERROR 6370: Only table or non-prejoin projection is allowed in FROM clause of aggregate projection

ERROR 6371: Only table or non-prejoin projection is allowed in FROM clause of top-k projection

ERROR 6372: ORDER BY is not allowed in aggregate projection. The aggregate projection is automatically ordered on group by columns

ERROR 6373: ORDER BY is not allowed in top-k projections. The top-k projection is automatically ordered on partition by columns

ERROR 6379: PARTITION BEST can only be used with single-phase user defined transform functions

ERROR 6381: PARTITION NODES can only be used with single-phase user defined transform functions

ERROR 6394: SEGMENTED BY / UNSEGMENTED is not allowed in aggregate projection. The aggregate projection is automatically segmented on group by columns

ERROR 6395: SEGMENTED BY / UNSEGMENTED is not allowed in top-k projections. The top-k projection is automatically segmented on partition by columns

ERROR 6518: At most one projection offset number can be entered

ERROR 6539: User Defined Transform *string* returns a string on which collation can't be applied in non default locale

ERROR 6543: Grouping functions only allowed with aggregates

ERROR 6544: Multilevel Aggregates are not allowed in projection

ERROR 6562: Hint PROJS cannot be used with hint SKIP_PROJS for the same table instance. Hint PROJS will be ignored

ERROR 6563: Hint SKIP_PROJS cannot be used with hint PROJS for the same table instance. Hint SKIP_PROJS will be ignored

ERROR 6568: Invalid projection name in hint: *string*. The whole hint will be ignored

ERROR 6640: Argument of hint *string* must be a positive integer

ERROR 6654: Batch/Prepass may only be used in the over(...) clause of a user defined transform in a Live Aggregate Projection

ERROR 6686: Cannot specify LIMIT with OVER(...) clause with a user defined transform

ERROR 6696: Column "*string.string*" refers to table "*string*" which is out of scope

ERROR 6697: Columns/Expressions in the partition clause have to appear in the SELECT list

ERROR 6701: Constants with the same parameter hint must be identical

ERROR 6800: Hint *string* must have one argument

ERROR 6834: Invalid use of *_oidrefs* hint

ERROR 6853: Live aggregate projections may only be one of the following types: Group-by, Top-K, or UDT

ERROR 6878: No previously saved query to associate with

ERROR 6899: Parameters cannot have NULL values

ERROR 6969: Subqueries for aggregate projections need to contain User Defined Transforms

ERROR 6972: Syntax Error. Struct member '*string*' has missing base type

ERROR 7014: UD Parameters value length [*value*] is more than the max allowed length [*value*]

ERROR 7051: Unsupported hint for input expr

ERROR 7109: Parameter '*string*' is required

ERROR 7147: Option *string* is repeated. Please specify each option only once

ERROR 7153: Syntax Error. At least one parameter has to be specified

ERROR 7164: Cannot use ORC format with FLEX tables

ERROR 7228: Cannot use *string* compression with *string* files

ERROR 7229: Cannot use PARQUET format with FLEX tables

ERROR 7311: Column "*string*" of table "*string*" already has a SET USING definition

ERROR 7313: Column *string* in Table *string* does not have SET USING expression

ERROR 7327: SET USING cannot be specified for IDENTITY/AUTO_INCREMENT column "*string*" of table "*string*"

ERROR 7335: Table *string* does not have any column with a SET USING expression

ERROR 7391: Cannot use an UDSOURCE with *string* files

ERROR 7420: Expression list has more than one *string* column

ERROR 7488: Number of key/value pairs [*value*] exceeds the maximum allowed

ERROR 7538: Sequence manipulation functions not supported in INSERT SELECT with implicit column that has default query

ERROR 7744: Try local hint only supports SELECT queries without table join

ERROR 7746: Try local hint requires all nodes to be up

ERROR 7750: WHERE clause syntax is incompatible with "try local" hint, hint is ignored

ERROR 7755: This projection is not valid with "try_local" hint

ERROR 7864: Column "*string*" of table "*string*" already has a SET USING/DEFAULT definition

ERROR 7875: SET USING/DEFAULT cannot be specified for IDENTITY/AUTO_INCREMENT column "*string*" of table "*string*"

ERROR 7909: Columns/Expressions in PARTITION BY must be first and in the same order with columns in SELECT

ERROR 7943: Cannot modify column with SET USING expression "*string*"

ERROR 7965: At most one WHEN MATCHED clause is allowed in MERGE

ERROR 7966: At most one WHEN NOT MATCHED clause is allowed in MERGE

ERROR 7967: MERGE/INSERT has more expressions than target columns

ERROR 7968: MERGE/INSERT has more target columns than expressions

ERROR 7982: Cannot use expressions in OVER(...) clause

ERROR 7989: Invalid column in MERGE statement: *string*. The insert filter of a WHEN NOT MATCHED clause cannot reference the target table *string*

ERROR 8011: SELECT expressions must have column labels

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42602

This topic lists the errors associated with the SQLSTATE 42602.

SQLSTATE 42602 Description

ERRCODE_INVALID_NAME

Error messages associated with this SQLState

ERROR 2383: Cannot create projections due to naming conflicts with existing projections

ERROR 2398: Cannot determine the best encoding options for some columns in table *string.string* due to insufficient data

ERROR 3059: DEPRECATED syntax. Segment expression "*string*" is a projection column name, segmenting on attribute "*string*"*stringstringstring* instead

ERROR 3378: Failed to generate a unique relation or sequence name

ERROR 3674: Invalid identifier name (*value* octets) "*string*"

ERROR 3703: Invalid name syntax

ERROR 3747: Invalid savepoint identifier *string*

ERROR 4159: Non-ASCII characters in names are prohibited

ERROR 4267: Only table column names & filler column names can appear in the list

ERROR 4451: Projection "*string*" does not exist

ERROR 4506: Query weight must be positive

ERROR 5360: User "*string*" does not exist

ERROR 5403: User/role "*string*" already exists

ERROR 5569: Either column "*string*" does not exist or table alias "*string*" is not allowed in "WHEN MATCHED THEN UPDATE SET"

ERROR 5769: Cannot drop the main vertica license

ERROR 5968: No such license *string* to drop

ERROR 5970: Node *string* is not a control node

ERROR 6089: Unknown control node *string*

ERROR 6676: Cannot drop global heir user "*string*". Try changing / clearing the GlobalHeirUsername config parameter

ERROR 7117: Refresh table name error: *string*

ERROR 7893: Invalid model name

ERROR 7903: The new model name cannot have schema/catalog

ERROR 7960: Invalid output view name

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42611

This topic lists the errors associated with the SQLSTATE 42611.

SQLSTATE 42611 Description

ERRCODE_INVALID_COLUMN_DEFINITION

Error messages associated with this SQLState

ERROR 6099: Using LONG column '*string*' in a constraint

ERROR 6100: Using PARTITION expression that returns a *string* value

ERROR 6101: Using PARTITION expression that returns a LONG value

ERROR 6102: Using PARTITION expression that returns a LONG value: '*string*'

ERROR 7312: Column *string* does not exist in Table *string*

ERROR 7342: *string* expression for column "*string*" may not refer to itself

ERROR 7389: Cannot set *string* for column "*string*" since it is referenced in default expression of column "*string*"

ERROR 7783: Cannot set *string* for column "*string*" since it is referenced in *string* expression of column "*string*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42622

This topic lists the errors associated with the SQLSTATE 42622.

SQLSTATE 42622 Description

ERRCODE_NAME_TOO_LONG

Error messages associated with this SQLState

ERROR 2462: Cannot open FileColumn because path is too long *string*

ERROR 3507: Identifier "*string*" is *value* octets long. Maximum limit is *value* octets

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42701

This topic lists the errors associated with the SQLSTATE 42701.

SQLSTATE 42701 Description

ERRCODE_DUPLICATE_COLUMN

Error messages associated with this SQLState

ERROR 2629: Column "*string*" is already of type "*string*"

ERROR 2638: Column "*string*" specified more than once

ERROR 2654: Column *string* specified more than once

ERROR 2655: Column *string* specified more than once in options list

ERROR 2662: Column name "*string*" already exists

ERROR 2663: Column name "*string*" appears more than once in USING clause

ERROR 2664: Column name "*string*" does not exist

ERROR 3144: Duplicate column *string* in ORDER BY list

ERROR 3145: Duplicate column name

ERROR 3150: Duplicate projection column name (projection: *string*)

ERROR 3154: Duplicated parameter "*string*" in parameter list

ERROR 5450: View definition can not contain duplicate column names "*string*"

ERROR 5878: Failed to create table *string*: duplicate column name *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42702

This topic lists the errors associated with the SQLSTATE 42702.

SQLSTATE 42702 Description

ERRCODE_AMBIGUOUS_COLUMN

Error messages associated with this SQLState

ERROR 2604: Clause *string* "string" is ambiguous

ERROR 2671: Column reference "string" is ambiguous

ERROR 2681: Common column name "string" appears more than once in left table

ERROR 2682: Common column name "string" appears more than once in right table

ERROR 5904: Flex table "string" has no internal "string" column

ERROR 6144: Ambiguous column reference in projection

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42703

This topic lists the errors associated with the SQLSTATE 42703.

SQLSTATE 42703 Description

ERRCODE_UNDEFINED_COLUMN

Error messages associated with this SQLState

ERROR 2359: Cannot assign to field "string" of column "string" because there is no such column in data type *string*

ERROR 2625: Column "string" does not exist;
Vertica does not support 'SELECT <table_name> FROM <table_name>'

ERROR 2633: Column "string" named as primary key does not exist

ERROR 2634: Column "string" not found in data type *string*

ERROR 2635: Column "string" of relation "string" does not exist

ERROR 2636: Column "string" specified in USING clause does not exist in left table

ERROR 2637: Column "string" specified in USING clause does not exist in right table

ERROR 2639: Column "string"."string" does not exist as a projection column

ERROR 2643: Column *string* does not exist

ERROR 2644: Column *string* does not exist in table

ERROR 2651: Column *string* must be loaded or computed

ERROR 2656: Column *string.string* does not exist

ERROR 2870: Could not identify column "string" in record data type

ERROR 4450: Projection "string" does not contain column "string"

ERROR 6695: Column "string"."string" must have CHAR, VARCHAR, LONG VARCHAR, VARBINARY, LONG VARBINARY, or USER DEFINED type

ERROR 7009: Tokenizer must have a "string" column output for stemming

ERROR 7167: Column "string" does not exist on table "string"

ERROR 7367: After excluding columns, no input column remains

ERROR 7393: Column 'string' does not exist in 'string' table

ERROR 7563: The type of *string* column cannot be retrieved

ERROR 7959: Column [*string*] does not exist in relation [*string*]

ERROR 7984: Column [*string*] is duplicated in parameter *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42704

This topic lists the errors associated with the SQLSTATE 42704.

SQLSTATE 42704 Description

ERRCODE_UNDEFINED_OBJECT

Error messages associated with this SQLState

ERROR 2067: '*string*' is not a known granularity for audits.
string

ERROR 2068: '*string*' is not a known TM task.
string

ERROR 2070: '*string*' is not a valid granularity for *string*.
string

ERROR 2073: '*string*' is not supported by index tool

ERROR 2274: Bootstrap error (most likely in Bootstrap.cpp): Unregistered name *string*

ERROR 2275: Bootstrap error (most likely in Bootstrap.cpp): Unregistered oid *value*

ERROR 2710: Constraint "*string*" does not exist

ERROR 2711: Constraint "*string*" does not exist on table "*string*"

ERROR 3001: DDL statement interfered with *string.nextval*

ERROR 3256: Error reported by client: *string*

ERROR 3442: Found eligible *value* processes to invite, but no matching nodes in catalog

ERROR 3637: Invalid Component Name '*string*'

ERROR 3655: Invalid epoch range

ERROR 3698: Invalid mergeout task identifier (Possible values are: [*0*, *value*])

ERROR 3715: Invalid object name

ERROR 3748: Invalid scope in ANALYZE_WORKLOAD*string*: schema or table *string* was altered

ERROR 3749: Invalid scope in ANALYZE_WORKLOAD: schema or table *string* does not exist

ERROR 3756: Invalid Sub-Component Name '*string*'

ERROR 3769: Invalid TM operation

ERROR 3779: Invalid user ID: *value*

ERROR 3842: Language does not exist: *string*

ERROR 3855: Library "*string*" does not exist

ERROR 3862: Library with name '*string*' does not exist

ERROR 4046: Network Interface "*string*" does not exist

ERROR 4047: Network Interface "*string*" is setup on another node

ERROR 4101: No role "*string*" exists

ERROR 4109: No storages in the specified epoch range

ERROR 4110: No such node *string*

ERROR 4111: No such object

ERROR 4112: No such projection

ERROR 4113: No such projection '*string*'

Vertica Documentation

Vertica Error Messages

ERROR 4123: No user or role "*string*" exists

ERROR 4129: No value found for parameter "*string*"

ERROR 4130: No value found for parameter *value*

ERROR 4137: Node *string* does not exist

ERROR 4216: Object '*string*' is not a projection

ERROR 4217: Object '*string*' is not a table

ERROR 4218: Object '*string*' is not a table or projection

ERROR 4223: OID *value* is not a sequence

ERROR 4224: OID *value* is not a Table or a View

ERROR 4446: Profile "*string*" does not exist

ERROR 4447: Profile '*string*' does not exist

ERROR 4594: Resource pool "*string*" does not exist

ERROR 4596: Resource pool '*string*' does not exist

ERROR 4614: Role "*string*" does not exist

ERROR 4616: Role "*string*" not found

ERROR 4650: Schema "*string*" does not exist

ERROR 4656: Schema, table, or projection "*string*" does not exist.
string

ERROR 4697: Sequence "*string*" does not exist

ERROR 4713: Sequence with name '*string*' does not exist

ERROR 4806: Subnet "*string*" does not exist

ERROR 4926: The *string* "*string*" does not exist

ERROR 4928: The *string* ["*string*"] does not exist

ERROR 5105: Tuning rule "*string*" does not exist

ERROR 5108: Type "*string*" does not exist

ERROR 5109: Type "*string*" is only a shell

ERROR 5112: Type *string* is only a shell

ERROR 5115: Type with OID *value* does not exist

ERROR 5227: Unrecognized drop object type: *value*

ERROR 5362: User or Role "*string*" not found

ERROR 5365: User available location ["*string*"] does not exist on node ["*string*"]

ERROR 5446: View "*string*" does not exist

ERROR 5459: Window "*string*" does not exist

ERROR 5532: Can not find any eligible locations in tier *string*

ERROR 5585: Fault Group "*string*" does not exist

ERROR 5614: Library *string* does not exist

ERROR 5688: User Defined Type "*string*" does not exist

ERROR 5797: Could not find the JVM resource pool

ERROR 5913: HCatalog database *string* does not exist

ERROR 5931: Invalid Policy Name '*string*'

ERROR 5965: New node cannot be placed in a non-existent Fault Group "*string*"

ERROR 5969: No table or projection named *string* exists

ERROR 5974: Node doesn't exist
ERROR 5977: Object does not exist
ERROR 6071: Type *value* with *odbc_subtype value* is not supported
ERROR 6149: Authentication "*string*" does not exist
ERROR 6156: Can not use empty authentication
ERROR 6285: Index "*string*" does not exist
ERROR 6354: No auth "*string*" exists
ERROR 6358: No such table '*string*'
ERROR 6423: Target standby node "*string*" does not exist
ERROR 6567: Invalid input parameters.
string
ERROR 6887: OID *value* is of a type unsupported for re-parenting
ERROR 7101: Invalid role oid encountered
ERROR 7275: Registry::getSequenceCache failed to get Sequence Cache in planning phase for [*value*]
ERROR 7704: Blob does not exist
ERROR 7706: Cannot remove blob: Blob does not exist
ERROR 7897: Model "*string*" does not exist
ERROR 7901: Model with name '*string*' does not exist
ERROR 7902: The model file does not exist

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42710

This topic lists the errors associated with the SQLSTATE 42710.

SQLSTATE 42710 Description

ERRCODE_DUPLICATE_OBJECT

Error messages associated with this SQLState

ERROR 2101: A sequence named "*string*" already exists
ERROR 2105: A table named "*string*" already exists
ERROR 2107: A view named "*string*" already exists

Vertica Documentation

Vertica Error Messages

ERROR 2273: Bootstrap error (most likely in Bootstrap.cpp): Oid *value* is already registered

ERROR 2276: Bootstrap error (most likely in Bootstrap.cpp):Name *string* is already registered

ERROR 2713: Constraint *string* already exists

ERROR 3153: Duplicated local temp table found in design queries: *string*

ERROR 3327: Existing object "*string*" is not a view

ERROR 3881: Location [*string*] already exists for node *string*

ERROR 4043: Network Interface "*string*" already exists

ERROR 4135: Node "*string*" already exists

ERROR 4213: Object "*string*" already exists

ERROR 4445: Profile "*string*" already exists

ERROR 4482: Projection with base name "*string*" already exists

ERROR 4564: Relation "*string*" already exists

ERROR 4565: Relation "*string*" already exists in schema "*string*"

ERROR 4593: Resource pool "*string*" already exists

ERROR 4621: Role "*string*" already exists

ERROR 4804: Subnet "*string*" already exists

ERROR 4805: Subnet "*string*" already exists for [*string*]

ERROR 5582: Fault Group "*string*" already exists

ERROR 5584: Fault Group "*string*" cannot depend on itself directly or indirectly

ERROR 5615: Location [*string*] conflicts with existing location [*string*] on node *string*

ERROR 5623: Network Interface "*string*" already exists for [*string*]

ERROR 6009: Resource pool *string* already exists

ERROR 6112: Unable to guarantee the same base name for all segmented buddy projections

ERROR 6148: Authentication "*string*" already exists

ERROR 6157: Can't create an index in schema "*string*": Schema cannot be found

ERROR 6158: Can't create an index named "*string*": Object already exists

ERROR 6188: Client authentication "*string*" already exists

ERROR 6508: Cannot move multiple relations with the same name "*string*" to schema "*string*"

ERROR 6735: Directed query with identifier "*string*" already exists

ERROR 6736: Directed query with identifier "*string*" does not exist

ERROR 7059: User "*string*" already exists

ERROR 7138: View "*string*" already exists in schema "*string*"

ERROR 7713: Error in blob creation: A blob with name '*string*' already exists

ERROR 7895: Model "*string*" already exists

ERROR 7896: Model "*string*" already exists in schema "*string*"

ERROR 7963: Relation [*string*] specified in parameter 'output_view' already exists

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42712

This topic lists the errors associated with the SQLSTATE 42712.

SQLSTATE 42712 Description

ERRCODE_DUPLICATE_ALIAS

Error messages associated with this SQLState

ERROR 4901: Table name "*string*" specified more than once

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42723

This topic lists the errors associated with the SQLSTATE 42723.

SQLSTATE 42723 Description

ERRCODE_DUPLICATE_FUNCTION

Error messages associated with this SQLState

ERROR 2278: Built-in function with the same name already exists: *string*

ERROR 3472: Function with same name and number of parameters already exists: *string*

ERROR 4220: Object with same name and number of parameters already exists: *string*

ERROR 4428: Procedure/Function with same name and number of parameters already exists in schema *string*

ERROR 4429: Procedure/Function with same name and number of parameters already exists: *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42725

This topic lists the errors associated with the SQLSTATE 42725.

SQLSTATE 42725 Description

ERRCODE_AMBIGUOUS_FUNCTION

Error messages associated with this SQLState

ERROR 3459: Function *string* is not unique

ERROR 4289: Operator is not unique: *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42803

This topic lists the errors associated with the SQLSTATE 42803.

SQLSTATE 42803 Description

ERRCODE_GROUPING_ERROR

Error messages associated with this SQLState

ERROR 2134: Aggregate function calls in subqueries cannot refer to columns in parent (outer) query

ERROR 2135: Aggregate function calls may not be nested
ERROR 2140: Aggregates not allowed in GROUP BY clause
ERROR 2141: Aggregates not allowed in JOIN conditions
ERROR 2142: Aggregates not allowed in WHERE clause
ERROR 2219: Argument *string* must not contain aggregates
ERROR 2543: Cannot use aggregate function in EXECUTE parameter
ERROR 2544: Cannot use aggregate function in function expression in FROM
ERROR 2640: Column "*string.string*" must appear in the GROUP BY clause or be used in an aggregate function
ERROR 4300: ORDER/GROUP BY expression not found in targetlist
ERROR 4634: Rule WHERE condition may not contain aggregate functions
ERROR 4667: SEGMENTED BY expression may not contain aggregate functions
ERROR 4841: Subquery uses ungrouped column "*string.string*" from outer query
ERROR 6139: All non-aggregate expressions in the projection SELECT list must appear in the GROUP BY clause
ERROR 6187: Cannot use aggregate expressions in the projection SELECT list without a GROUP BY clause
ERROR 6277: Grouping functions not allowed in GROUP BY clause
ERROR 6278: Grouping functions not allowed in JOIN conditions
ERROR 6279: Grouping functions not allowed in WHERE clause
ERROR 7085: Aggregate function calls cannot contain grouping functions
ERROR 7099: Grouping functions cannot be nested
ERROR 7182: Grouping function arguments need to be group by expressions

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42804

This topic lists the errors associated with the SQLSTATE 42804.

SQLSTATE 42804 Description

ERRCODE_DATATYPE_MISMATCH

Error messages associated with this SQLState

ERROR 2217: Argument *string* must be type float, not type *string*

ERROR 2218: Argument *string* must be type integer, not type *string*

ERROR 2222: Argument *string* must not return a set

ERROR 2224: Argument of *string* must be type boolean, not type *string*

ERROR 2225: Argument of *string* must not return a set

ERROR 2231: Array assignment requires type *string* but expression is of type *string*

ERROR 2232: Array assignment to "*string*" requires type *string* but expression is of type *string*

ERROR 2234: Array subscript must have type integer

ERROR 2358: Cannot assign to field "*string*" of column "*string*" because its type *string* is not a composite type

ERROR 2527: Cannot subscript type *string* because it is not an array

ERROR 2630: Column "*string*" is of type *string* but default expression is of type *string*

ERROR 2631: Column "*string*" is of type *string* but expression is of type *string*

ERROR 2846: Could not determine actual result type for function "*string*" declared to return type *string*

ERROR 2850: Could not determine row description for function returning record

ERROR 3429: For '*string*', types *string* and *string* are inconsistent

ERROR 3447: Function "*string*" in FROM has unsupported return type *string*

ERROR 3545: Index expression may not return a set

ERROR 3801: IS DISTINCT FROM requires = operator to yield boolean

ERROR 3943: Mismatched types in VALUES LESS THAN expressions

ERROR 4069: No column alias was provided

ERROR 4199: Number of aliases does not match number of columns

ERROR 4284: Operator *string* must not return a set

ERROR 4285: Operator *string* must return type boolean, not type *string*

ERROR 4317: Parameter *\$value* of type *string* cannot be coerced to the expected type *string*

ERROR 4625: Row comparison operator must not return a set

ERROR 4626: Row comparison operator must yield type boolean, not type *string*

ERROR 4803: Subfield "*string*" is of type *string* but expression is of type *string*

ERROR 6480: Column "*string*" is of type *string* but expression in Access Policy is of type *string*. It will be coerced at execution time

ERROR 7221: Argument of type *string* must be type boolean, not *string*

ERROR 7222: Argument of type *string* must be type boolean, not type *string*

ERROR 7309: Column "*string*" is of type *string* but set using expression is of type *string*

ERROR 7708: Column must be a string type

ERROR 7929: Column to be added/refreshed is of type *string* but default/set-using expression is of type *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42809

This topic lists the errors associated with the SQLSTATE 42809.

SQLSTATE 42809 Description

ERRCODE_WRONG_OBJECT_TYPE

Error messages associated with this SQLState

- ERROR 2037: *string* is not a supported analytic function
- ERROR 2131: Aggregate function calls cannot contain analytic function calls
- ERROR 2132: Aggregate function calls cannot contain sequence function calls
- ERROR 2668: Column notation *.string* applied to type *string*, which is not a composite type
- ERROR 2755: COPY requires relation *string* to be a Table, not a *string*
- ERROR 2810: Could not add location [*string*]: Directory not empty
- ERROR 2811: Could not add location [*string*]: Not a directory
- ERROR 3114: DISTINCT specified, but *string* is not an aggregate function
- ERROR 3421: First argument to modularhash_wrapper must be an integer constant
- ERROR 3422: First argument to modularhash_wrapper must be of type integer, not *string*
- ERROR 3463: Function *string(string)* is not an aggregate
- ERROR 3552: Inherited relation "*string*" is not a table
- ERROR 3669: Invalid function given
- ERROR 3965: modularhash_wrapper must have two arguments: an integer constant and a call to modularhash_internal
- ERROR 3966: modularhash_wrapper second argument is not modularhash_internal or a constant
- ERROR 4215: Object "*string*" is not a projection
- ERROR 4270: Op ANY/ALL (array) requires array on right side
- ERROR 4271: Op ANY/ALL (array) requires operator not to return a set
- ERROR 4272: Op ANY/ALL (array) requires operator to yield boolean
- ERROR 4542: Record type has not been registered
- ERROR 4657: Second argument to *string* must be a non-negative integer constant
- ERROR 4931: The argument to *string* cannot be null
- ERROR 4932: The argument to *string* must be a constant
- ERROR 4987: Third argument to *string* must be a constant

ERROR 5111: Type *string* is not composite

ERROR 6036: Table "*string*" is not a flex table

ERROR 6691: CAST and current_date/current_time functions are not supported as analytic functions

ERROR 6951: Skip lazy projection creation for table *string.string* since the object referenced by the hint is either invalid or of the wrong type (expected *string*)

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42830

This topic lists the errors associated with the SQLSTATE 42830.

SQLSTATE 42830 Description

ERRCODE_INVALID_FOREIGN_KEY

Error messages associated with this SQLState

ERROR 3438: Foreign keys not specified

ERROR 3531: Incompatible data types between primary and foreign key columns: fk: *string*, pk: *string*

ERROR 4207: Number of primary and foreign keys must be the same

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42846

This topic lists the errors associated with the SQLSTATE 42846.

SQLSTATE 42846 Description

ERRCODE_CANNOT_COERCE

Error messages associated with this SQLState

ERROR 2015: *string* could not convert type *string* to *string*

ERROR 2366: Cannot cast type *string* to *string*

ERROR 4986: Third argument of *string* could not be converted from type *string* to type *string*

ERROR 6510: Column "*string*" is of type *string* but expression in Access Policy is of type *string*. Cannot coerce expression

ERROR 7310: Column "*string*" is of type *string* but the *string* expression is of type *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42883

This topic lists the errors associated with the SQLSTATE 42883.

SQLSTATE 42883 Description

ERRCODE_UNDEFINED_FUNCTION

Error messages associated with this SQLState

ERROR 2126: Aggregate *string(string)* does not exist

ERROR 2127: Aggregate *string(*)* does not exist

ERROR 3456: Function *string* does not exist

ERROR 3457: Function *string* does not exist, or permission is denied for *string*

ERROR 3462: Function *string* with the specified arguments does not exist

ERROR 3930: Meta-function *string* cannot be used in COPY

ERROR 3931: Meta-function *string* cannot be used in INSERT

ERROR 3932: Meta-function *string* cannot be used in UPDATE

ERROR 3933: Meta-function *string* cannot be used with FROM

ERROR 3934: Meta-function ("*string*") can be used only in the Select clause

ERROR 4067: No binary input function available for type *string*

ERROR 4068: No binary output function available for type *string*

ERROR 4083: No input function available for type *string*

ERROR 4091: No output function available for type *string*

ERROR 4286: Operator does not exist: *string*

ERROR 4290: Operator requires run-time type coercion: *string*

ERROR 5394: User procedure call (*value*) is not supported with FROM

ERROR 5910: Function *string* with the specified type and arguments does not exist

ERROR 6333: Meta-functions or non-deterministic functions cannot be used in projection column expressions

ERROR 6809: Inappropriate usage of meta-function ("*string*")

ERROR 7323: Meta-functions cannot be used in *string* expressions

ERROR 7324: Meta-functions cannot be used in *string* query definitions

ERROR 7340: VOLATILE functions cannot be used in a *string* expression when adding a column

ERROR 7341: VOLATILE functions cannot be used in a *string* query when adding a column

ERROR 7589: VOLATILE functions cannot be used in a *string* expression

ERROR 7590: VOLATILE functions cannot be used in a *string* query

ERROR 7657: Function "*string*" does not exist

ERROR 7688: Calling metafunctions '*string*' and '*string*' within the same query is not supported

ERROR 7692: Multiple calls to metafunction '*string*' within the same query are not supported

ERROR 7970: Meta-function *string* cannot be used in MERGE/UPDATE

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42939

This topic lists the errors associated with the SQLSTATE 42939.

SQLSTATE 42939 Description

ERRCODE_RESERVED_NAME

Error messages associated with this SQLState

ERROR 2297: Can not drop default profile
ERROR 2299: Can not rename default profile
ERROR 2418: Cannot drop role "string"
ERROR 2488: Cannot rename role *string*
ERROR 2489: Cannot rename system column epoch
ERROR 2665: Column name "string" is reserved
ERROR 2666: Column name *string* is reserved
ERROR 3778: Invalid use of reserved the column name "string"
ERROR 4030: Names starting with "v_" are reserved names
ERROR 4953: The name "string" is a reserved name
ERROR 4962: The prefix "sys_" is reserved for system tuning rule
ERROR 6195: Column name "string" is reserved in aggregate projections
ERROR 7441: Invalid use of reserved column name "string"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42P20

This topic lists the errors associated with the SQLSTATE 42P20.

SQLSTATE 42P20 Description

ERRCODE_WINDOWING_ERROR

Error messages associated with this SQLState

ERROR 2011: *string* cannot use the WITHIN GROUP clause
ERROR 2041: *string* may only have one sort expression in the WITHIN GROUP clause
ERROR 2043: *string* must contain an ORDER BY clause within its analytic clause
ERROR 2044: *string* must NOT contain an ORDER BY clause or WINDOWING clause within its analytic clause
ERROR 2045: *string* must NOT contain WINDOWING clause within its analytic clause

Vertica Documentation

Vertica Error Messages

ERROR 2047: *string* only supports the Integer, Float, Interval and Numeric data types

ERROR 2182: Analytic functions are allowed only in a SELECT list and/or ORDER BY clause

ERROR 2185: Analytic functions are not supported in the PARTITION BY of an OVER clause

ERROR 2188: Analytic functions must have a FROM clause

ERROR 2189: Analytic functions not allowed in *string*

ERROR 2305: Can't cast the window bound into Int

ERROR 2306: Can't cast the window bound into the same data type of the ORDER BY column

ERROR 2465: Cannot override ORDER BY clause of window "*string*"

ERROR 2466: Cannot override PARTITION BY clause of window "*string*"

ERROR 2524: Cannot specify frame clause of window "*string*"

ERROR 3435: For range moving window, OrderBy expression must be one of Int, Float, Time, Timestamp, Interval, Date or Numeric

ERROR 3446: Frame clause not allowed without windowing order by

ERROR 3839: Keyword "ALL" is invalid in analytic functions

ERROR 4362: PERCENTILE_CONT/PERCENTILE_DISC must have the WITHIN GROUP clause

ERROR 4363: PERCENTILE_CONT/PERCENTILE_DISC must NOT contain an ORDER BY clause or WINDOWING clause within its analytic clause

ERROR 4811: Subqueries are not supported in the PARTITION BY of a timeseries OVER clause

ERROR 5006: Time Series Aggregate Functions are not supported in the PARTITION BY of a timeseries OVER clause

ERROR 5010: Time Series timestamp alias/Time Series Aggregate Functions not allowed in *string*

ERROR 5460: Window "*string*" is already defined

ERROR 5461: Window frame cannot end with PRECEDING if start is CURRENT ROW

ERROR 5462: Window frame cannot end with PRECEDING or CURRENT ROW if start is FOLLOWING

ERROR 5463: Window frame cannot end with UNBOUNDED PRECEDING

ERROR 5464: Window frame cannot start with UNBOUNDED FOLLOWING

ERROR 5466: Window frame logical offset must be a non-negative number to be consistent with the sort column type

ERROR 5467: Window frame logical offset must be an Interval (Day to Second or Year to Month) to be consistent with the sort column type

ERROR 5468: Window frame logical offset must be an Interval (Day to Second) to be consistent with the sort column type

ERROR 5469: Window frame logical offset must be an interval to be consistent with the sort column type

ERROR 5470: Window frame logical offset must be Int when the sort column type is Int

ERROR 5471: Window frame logical offset must be the same type as the sort column type (Interval Day to Second)

ERROR 5472: Window frame logical offset must be the same type as the sort column type (Interval Year to Month)

ERROR 5473: Window frame logical or physical offset must be a constant

ERROR 5474: Window frame logical or physical offset must be non-negative number or interval

ERROR 5475: Window frame physical offset must be non-negative number

ERROR 5477: Window ordering clause can only contain a single sort key if RANGE is used

ERROR 5478: Windowing not supported for User Defined Analytic functions

ERROR 6276: Grouping functions cannot appear within Analytic functions

ERROR 6900: Partition Prepass/Batch may only be used with Live Aggregate Projection

ERROR 7838: Windowing not supported for analytic usage of BOOL_XOR

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V01

This topic lists the errors associated with the SQLSTATE 42V01.

SQLSTATE 42V01 Description

ERRCODE_UNDEFINED_TABLE

Error messages associated with this SQLState

ERROR 2308: Can't find anchor table

ERROR 2312: Can't find table

ERROR 2313: Can't find table "*string*"

ERROR 2714: Constraint *string* does not exist

ERROR 2948: CTAS: table "*string*" was dropped in another session (DDL interference)

ERROR 3642: Invalid CTAS query: *string*

ERROR 3760: Invalid table name

ERROR 3761: Invalid table name "*string*"

ERROR 3762: Invalid table name *string*

ERROR 3953: Missing FROM-clause entry for table "*string*"

ERROR 3954: Missing FROM-clause entry in subquery for table "*string*"

ERROR 4416: Primary table "*string*" does not exist

ERROR 4566: Relation "*string*" does not exist

ERROR 4567: Relation "*string*" in FOR UPDATE clause not found in FROM clause

ERROR 4568: Relation "*string.string*" does not exist

ERROR 4570: Relation with OID *value* does not exist

ERROR 4883: Table "*string.string*" does not exist

ERROR 4898: Table does not exist (oid=*value*)

ERROR 4911: Table with OID *value* does not exist

ERROR 4912: Table/View with name '*string*' does not exist

ERROR 6220: CREATE INDEX failed
ERROR 6221: DDL interfered with this statement; table concurrently dropped
ERROR 6300: Invalid text-index parameters
ERROR 6418: Table "*string*" no longer exists
ERROR 6420: Table '*stringstringstring*' does not exist
ERROR 6729: DDL Interfered! table "*string*" does not exist
ERROR 6730: DDL statement interfered with this statement; table was dropped
ERROR 7285: Table "*string*" does not exist in HCatalog schema *string*
ERROR 7438: Invalid table name: *string*
ERROR 7529: Relation '*string*' does not exist
ERROR 7847: Relation [*string*] does not exist or access was denied

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V02

This topic lists the errors associated with the SQLSTATE 42V02.

SQLSTATE 42V02 Description

ERRCODE_UNDEFINED_PARAMETER

Error messages associated with this SQLState

ERROR 3638: Invalid configuration parameter *string*; aborting configuration change
ERROR 4321: Parameter *value* is not set
ERROR 4984: There is no parameter *\$value*
ERROR 5202: Unknown configuration parameter
ERROR 6201: Configuration parameter *string* cannot be set at this level; aborting configuration change
ERROR 6511: Configuration parameter *string* cannot be set at this level; no value to return
ERROR 7350: --init_method not allowed with argument --initial_centers_table
ERROR 7357: A problem occurred during the execution of a detect_outliers iteration.
Detail: *string*
ERROR 7358: A problem occurred during the execution of a MinMax iteration.
Detail: *string*

Vertica Documentation

Vertica Error Messages

- ERROR 7360: A problem occurred during the execution of a robust_zscore iteration.
Detail: *string*
- ERROR 7361: A problem occurred during the execution of a zscore iteration.
Detail: *string*
- ERROR 7363: A problem occurred during the execution of balance.
Detail: *string*
- ERROR 7364: A problem occurred during the execution of outlier detection.
Detail: *string*
- ERROR 7437: Invalid parameter: *string*
- ERROR 7494: Parsing of the optional arguments.
Detail: *string*
- ERROR 7806: Parsing of the optional arguments.
Details: *string*
- ERROR 7891: init_method not allowed with argument initial_centers_table
- ERROR 7947: A problem occurred during table
renaming.
Detail: *string*
- ERROR 7950: A problem occurred during the execution of
a mode imputation with partition by.
Detail: *string*
- ERROR 7951: A problem occurred during the execution of
a mode imputation without partition by.
Detail: *string*
- ERROR 7952: A problem occurred during the execution of
computing mode for the first MiniBatch.
Detail: *string*
- ERROR 7953: A problem occurred during the execution of
computing mode.
Detail: *string*
- ERROR 7955: A problem occurred during the execution of
MiniBatch Table Merging.
Detail: *string*
- ERROR 7956: A problem occurred during the execution of a mean
imputation without partition by.
Detail: *string*
- ERROR 7957: A problem occurred during the execution of a mean
imputation with partition by.
Detail: *string*
- ERROR 7958: A problem occurred during the execution of a mean
imputation with partition by.
Detail: *string*
- ERROR 8017: Type mismatch of an optional argument: *string* must be a valid integer
- ERROR 8018: Type mismatch of an optional argument: *string* must be a valid numeric
- ERROR 8019: Type mismatch of an optional argument: *string* must be a valid string
- ERROR 8035: A problem occurred during the execution
of DROP TABLE impute_201701_80425147_p1.
Detail: *string*
- ERROR 8036: A problem occurred during the execution of
DROP TABLE impute_201701_80425147_p1,impute_201701_80425147_p2.
Detail: *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V03

This topic lists the errors associated with the SQLSTATE 42V03.

SQLSTATE 42V03 Description

ERRCODE_DUPLICATE_CURSOR

Error messages associated with this SQLState

ERROR 2615: Closing existing cursor "*string*"

ERROR 2968: Cursor "*string*" already exists

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V04

This topic lists the errors associated with the SQLSTATE 42V04.

SQLSTATE 42V04 Description

ERRCODE_DUPLICATE_DATABASE

Error messages associated with this SQLState

ERROR 2706: Connection to database [*string*] already exists

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V06

This topic lists the errors associated with the SQLSTATE 42V06.

SQLSTATE 42V06 Description

ERRCODE_DUPLICATE_SCHEMA

Error messages associated with this SQLState

ERROR 4649: Schema "*string*" already exists

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V07

This topic lists the errors associated with the SQLSTATE 42V07.

SQLSTATE 42V07 Description

ERRCODE_DUPLICATE_TABLE

Error messages associated with this SQLState

ERROR 4753: Skip lazy projection creation since super projection for table *string.string* already exists

ERROR 6952: Skip lazy projection creation since a projection enforcing key constraint '*string*' for table *string.string* already exists

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V08

This topic lists the errors associated with the SQLSTATE 42V08.

SQLSTATE 42V08 Description

ERRCODE_AMBIGUOUS_PARAMETER

Error messages associated with this SQLState

ERROR 2848: Could not determine data type of parameter *\$value*

ERROR 3534: Inconsistent types deduced for parameter *\$value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V09

This topic lists the errors associated with the SQLSTATE 42V09.

SQLSTATE 42V09 Description

ERRCODE_AMBIGUOUS_ALIAS

Error messages associated with this SQLState

ERROR 4908: Table reference "*string*" is ambiguous

ERROR 4909: Table reference *value* is ambiguous

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V10

This topic lists the errors associated with the SQLSTATE 42V10.

SQLSTATE 42V10 Description

ERRCODE_INVALID_COLUMN_REFERENCE

Error messages associated with this SQLState

ERROR 2046: *string* not allowed in *string* clause

ERROR 2050: *string* position *value* is not in select list

ERROR 2221: Argument *string* must not contain variables

ERROR 3467: Function expression in FROM may not refer to other relations of same query level

ERROR 3820: JOIN/ON clause refers to "*string*", which is not part of JOIN

ERROR 4832: Subquery in FROM may not refer to other relations of same query level

ERROR 4877: Table "*string*" has *value* columns available but *value* columns specified

ERROR 5057: Too many column aliases specified for function *string*

ERROR 5194: UNION/INTERSECT/EXCEPT member statement may not refer to other relations of same query level

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V11

This topic lists the errors associated with the SQLSTATE 42V11.

SQLSTATE 42V11 Description

ERRCODE_INVALID_CURSOR_DEFINITION

Error messages associated with this SQLState

ERROR 2522: Cannot specify both SCROLL and NO SCROLL

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V13

This topic lists the errors associated with the SQLSTATE 42V13.

SQLSTATE 42V13 Description

ERRCODE_INVALID_FUNCTION_DEFINITION

Error messages associated with this SQLState

ERROR 2038: *string* is not a supported Time Series Aggregate Function

ERROR 2139: Aggregates may not return sets

ERROR 2173: An error occurred on node [*string*] when setting up the function [*string*]: [*string*]

ERROR 2177: An error occurred when setting up function "*string*"

ERROR 2397: Cannot determine result data type

Vertica Documentation

Vertica Error Messages

ERROR 2451: Cannot load data from 0 sources; please specify 1 or more (on node [string])

ERROR 2494: Cannot RETURNREJECTED with multiple files or data sources

ERROR 3113: DISTINCT is supported only for single-argument aggregates

ERROR 3476: Functions in language *string* can be created only in fenced mode

ERROR 3604: Interpolation scheme *string* for Time Series Aggregate Function *string* is not supported

ERROR 3708: Invalid null argument for TSA function *string*

ERROR 3843: Language(*string*) does not match the language associated with the library(*string*)

ERROR 3854: Length of a string in a return type must be greater than zero

ERROR 3860: Library file is not loaded

ERROR 3861: Library not found: *string*

ERROR 3929: Meta functions cannot be used in UDX definitions

ERROR 4086: No language specified

ERROR 4095: No procedure source specified

ERROR 4096: No procedure user specified

ERROR 4251: Only one expression is allowed

ERROR 4257: Only simple "RETURN expression" is allowed

ERROR 4409: Precision of a numeric in a return type must be greater than zero

ERROR 4608: Return type *string* is not supported for SQL functions

ERROR 4609: Return type mismatch in a function declared to return *string*

ERROR 4610: Return type mismatch in function declared to return *string*

ERROR 4746: Setting up function "*string*" failed

ERROR 4794: Strictness in the DDL and the function factory class don't match. Function was not created

ERROR 4858: Syntax error in syntax definition at offset *value*

ERROR 4949: The interpolation argument for Time Series Aggregate Function *string* must be a constant string

ERROR 5457: Volatility in the DDL and the function factory class don't match. Function was not created

ERROR 5476: Window functions cannot return sets

ERROR 5777: Cannot set up function [string] on node: *string*

ERROR 6072: UDFFileSystem only supports C++ unfenced mode

ERROR 6536: The `getPrototype(value)` and `getReturnType(value)` method must have matching number of returnTypes

ERROR 6537: The return types declared by `getPrototype()` and `getReturnType()` do not match

ERROR 6850: Library [string] built with incompatible SDK version [string]. Please rebuild with SDK version [string] and recreate the library

ERROR 6999: The library [string] for the function [string] was compiled with an incompatible SDK Version [string]

ERROR 7270: Invalid schema.table.column, table.column, or column specification

ERROR 7830: Only MIN/MAX and AND/OR are allowed to use DISTINCT

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V15

This topic lists the errors associated with the SQLSTATE 42V15.

SQLSTATE 42V15 Description

ERRCODE_INVALID_SCHEMA_DEFINITION

Error messages associated with this SQLState

ERROR 2470: Cannot plan query because no super projections are safe, some node(s) are down
ERROR 2945: CREATE specifies a schema (*string*) different from the one being created (*string*)
ERROR 3365: Failed to create default projections for table "*string*".*string*: *string*
ERROR 3586: Insufficient projections to answer query
ERROR 4097: No projections eligible to answer query
ERROR 4878: Table "*string*" has an out-of-date super projection "*string*"
ERROR 6774: Failed to create default key projections for table "*string*".*string*: *string*
ERROR 7126: Table "*string*" (non-anchor in prejoin) has an out-of-date prejoin projection "*string*"
ERROR 7142: Failed to create hcatalog schema: *string*
ERROR 7876: Table "*string*" has an out-of-date projection "*string*"

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V16

This topic lists the errors associated with the SQLSTATE 42V16.

SQLSTATE 42V16 Description

ERRCODE_INVALID_TABLE_DEFINITION

Error messages associated with this SQLState

ERROR 2104: A table cannot have only IDENTITY/AUTO-INCREMENT columns

ERROR 2420: Cannot drop the constraint. (Table "*string*" has a foreign key constraint referencing the specified primary key constraint)

ERROR 2421: Cannot drop the constraint. (There is at least one prejoin projection dependent on the specified foreign key constraint)

ERROR 2622: Column "*string*" cannot be declared SETOF

ERROR 2626: Column "*string*" from table "*string*" in the SEGMENTED BY expression is required to be present in the projection, but is not

ERROR 2712: Constraint "*string*" for relation "*string*" already exists

ERROR 3508: IDENTITY/AUTO-INCREMENT columns are not allowed in temporary tables

ERROR 3874: Local temporary table constraint cannot reference a non-local table

ERROR 3901: MATCH types other than SIMPLE (the default) are not supported for foreign key constraints

ERROR 3987: Multiple primary keys for table "*string*" are not allowed

ERROR 4162: Non-local table constraint cannot reference a local temporary table

ERROR 4229: ON DELETE actions other than NO ACTION are not supported for foreign key constraints

ERROR 4234: ON UPDATE actions other than NO ACTION are not supported for foreign key constraints

ERROR 4413: Primary constraint for relation "*string*" already exists

ERROR 4415: Primary keys not specified

ERROR 4469: Projection anchor table is not partitioned

ERROR 4550: Referenced primary key constraint does not exist

ERROR 4876: Table "*string*" does not exist

ERROR 4881: Table "*string*" is not partitioned

ERROR 4899: Table is not partitioned

ERROR 4900: Table must have at least one column

ERROR 5269: Unsupported constraint type

ERROR 5548: Constraint not supported for user defined type column *string*

ERROR 5552: Correlation constraint not supported for user defined types

ERROR 5874: Failed to add table *string* of hcatalog schema *string* to catalog: no columns

ERROR 5876: Failed to alter table *string* of hcatalog schema *string* to catalog: no columns

ERROR 5879: Failed to describe hcatalog table

ERROR 5948: Local temporary objects may not specify a schema name

ERROR 6171: Cannot merge multiple partitions on an aggregate projection

ERROR 6396: SEGMENTED BY expression contains expressions not present in the SELECT list

ERROR 6434: TM task is not applicable to projections with aggregates

ERROR 6839: Key constraints on an external table cannot be enabled or disabled
ERROR 7122: Staging table's partition key and target table's partition key have different datatype
ERROR 7233: Check constraint '*string*' does not reference any columns in the table
ERROR 7234: Check constraint '*string*': references to column '*string*' are not allowed in check predicates
ERROR 7235: Check constraint '*string*': subqueries are not allowed in check predicates
ERROR 7236: Check constraint predicate is too long; *value* bytes, maximum length is *value*
ERROR 7237: Check constraints on an external table cannot be enabled or disabled
ERROR 7238: Column "*string*" referenced by check constraint '*string*' is not in the table
ERROR 7260: Function "*string*" referenced by check constraint '*string*' is a meta-function
ERROR 7261: Function "*string*" referenced by check constraint '*string*' is an aggregate function
ERROR 7262: Function "*string*" referenced by check constraint '*string*' is not immutable
ERROR 7263: Function "*string*" referenced by check constraint '*string*' returns a set rather than a single value

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V17

This topic lists the errors associated with the SQLSTATE 42V17.

SQLSTATE 42V17 Description

ERRCODE_INVALID_OBJECT_DEFINITION

Error messages associated with this SQLState

ERROR 2387: Cannot create projections involving external table *string*
ERROR 3075: Design type *string* is invalid
ERROR 3078: Optimization objective *string* is invalid
ERROR 3199: Error during deployment querying deployment projections table for workspace *string*
ERROR 3200: Error during deployment querying design projections table for design *string* in workspace *string*
ERROR 3201: Error during deployment while querying deployment projections table for workspace *string*
ERROR 3204: Error during drop design from deployment for workspace *string*
ERROR 3206: Error during extend catalog while querying deployments table for workspace *string*
ERROR 3207: Error during getDesignTablesFromDeployment in workspace *string*

Vertica Documentation

Vertica Error Messages

ERROR 3213: Error during remove deployment drops from deployment *string* for workspace *string*

ERROR 3227: Error in querying *string.string*

ERROR 3269: Error while checking whether there are only incremental design deployed for deployment *string* in workspace *string*

ERROR 3271: Error while querying designs table for workspace *string*

ERROR 3968: More than one IDENTITY/AUTO_INCREMENT column defined for table "*string*"

ERROR 3983: Multiple instances of deployment *string* in workspace *string*

ERROR 4128: No valid projections found

ERROR 4230: ON DELETE rule may not use NEW

ERROR 4231: ON INSERT rule may not use OLD

ERROR 4232: ON SELECT rule may not use NEW

ERROR 4233: ON SELECT rule may not use OLD

ERROR 4635: Rule WHERE condition may not contain references to other relations

ERROR 4636: Rules with WHERE conditions may only have SELECT, INSERT, UPDATE, or DELETE actions

ERROR 4919: Temporary table projections are not allowed for this operation

ERROR 4982: There is no deployment *string* in workspace *string*

ERROR 4989: This function cannot be called on design *string* located in design workspace *string*

ERROR 4990: This function cannot be called on design *string*, when its design mode is *string*

ERROR 5367: User defined aggregate must return exactly one column. Function *string* returns *value*

ERROR 5369: User defined analytic must return exactly one column

ERROR 5384: User defined transform must provide names or aliases for return columns

ERROR 5385: User defined transform must return at least one column

ERROR 5527: An error occurred on node *string* when setting up the type, message:
string

ERROR 5721: Purge is not allowed on temporary tables

ERROR 6095: UseLongStrings has been deprecated

ERROR 6166: Cannot create top-k projection: projection columns and limit are too big for the top-k buffer

ERROR 6483: Invalid enum value for parameter name

ERROR 6627: Aggregate projections may only contain User Defined Transforms with partition by BATCH in the outer query

ERROR 6628: Aggregate projections may only contain User Defined Transforms with partition by PREPASS or BATCH

ERROR 6659: Can not create prejoin projection

ERROR 6660: Can not create prejoin projection when node is recovering

ERROR 6970: Subqueries in aggregate projections may only contain User Defined Transforms with partition by PREPASS

ERROR 6993: The batch and prepass User Defined Transform Functions' signatures are not compatible for use in a live aggregate projection

ERROR 6994: The batch User Defined Transform Function does not have identical input and output signature

ERROR 7788: Only projections with single phase PREPASS User Defined Transforms can have user specified sort order or segmentation

ERROR 7789: ORDER BY / SEGMENTED BY / UNSEGMENTED are not allowed in projections with two phase User Defined Transforms. The projection is automatically ordered and segmented on partition by columns

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V18

This topic lists the errors associated with the SQLSTATE 42V18.

SQLSTATE 42V18 Description

ERRCODE_INDETERMINATE_DATATYPE

Error messages associated with this SQLState

ERROR 2847: Could not determine data type of column *\$value*

ERROR 3609: Interval must be single datetime field

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V21

This topic lists the errors associated with the SQLSTATE 42V21.

SQLSTATE 42V21 Description

ERRCODE_UNDEFINED_PROJECTION

Error messages associated with this SQLState

ERROR 2311: Can't find projection *value*

ERROR 2430: Cannot find projection column *value*
ERROR 3005: DDL statement interfered with refresh operation
ERROR 3736: Invalid projection name
ERROR 3737: Invalid projection name *string*
ERROR 4452: Projection "*string*" does not exist or was just dropped
ERROR 4474: Projection does not exist
ERROR 4905: Table or projection "*string*" does not exist
ERROR 5563: DDL statement interfered with this operation
ERROR 7528: Projecton "*string*" does not exist
ERROR 7863: Can't find projection: *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V24

This topic lists the errors associated with the SQLSTATE 42V24.

SQLSTATE 42V24 Description

ERRCODE_UNDEFINED_USER

Error messages associated with this SQLState

ERROR 7213: Current user no longer exists

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V25

This topic lists the errors associated with the SQLSTATE 42V25.

SQLSTATE 42V25 Description

ERRCODE_PATTERN_MATCH_ERROR

Error messages associated with this SQLState

ERROR 2227: Argument to `test_pattern_event_eval` must be > 0 and less than the total number of events

ERROR 2228: Argument to `test_pattern_event_eval` must be a constant

ERROR 2553: Cannot use more than one pattern

ERROR 2555: Cannot use pattern test functions with pattern match functions

ERROR 3025: Defining more than 52 events is not supported

ERROR 3288: Event "*string*" in PATTERN clause is not defined in the DEFINE clause

ERROR 3289: Event ANY_ROW cannot be used under `*`, `+`, `?`, or `|` when the select list contains the pattern function `event_name()`

ERROR 3290: Event ANY_ROW is a reserved event and cannot be user defined

ERROR 3294: Event expressions cannot contain analytic functions

ERROR 3295: Event expressions cannot contain correlated expressions

ERROR 3296: Event expressions cannot contain subqueries

ERROR 3297: Event name "*string*" defined more than once

ERROR 4353: Pattern events must be mutually exclusive

ERROR 4354: Pattern match query cannot contain having clause, group clause, aggregates, or distinct

ERROR 4355: Pattern match query cannot contain timeseries clause

ERROR 4356: Pattern matching recursion limit reached

ERROR 4358: `PatternMatchingMaxPartition` must be greater than 0

ERROR 4359: `PatternMatchingMaxPartitionMatches` must be greater than 0

ERROR 4360: `PatternMatchingPerMatchWorkspaceSize` must be greater than 0 and less than 1024

ERROR 4494: Queries with user-defined transform functions (*string*) cannot have a MATCH clause

ERROR 4507: Query with analytic function *string* cannot have a MATCH clause

ERROR 4509: Query with pattern matching function *string* must include a MATCH clause

ERROR 4605: RESULTS GROUPED BY MATCH is not supported

ERROR 5283: Unsupported pattern operator

ERROR 7607: Pattern matching DFA Algorithm work space size limit reached

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 42V26

This topic lists the errors associated with the SQLSTATE 42V26.

SQLSTATE 42V26 Description

ERRCODE_DUPLICATE_NODE

Error messages associated with this SQLState

ERROR 4058: New node matches existing node *string*

ERROR 4063: New values for node *string* matches existing node *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 53000

This topic lists the errors associated with the SQLSTATE 53000.

SQLSTATE 53000 Description

ERRCODE_INSUFFICIENT_RESOURCES

Error messages associated with this SQLState

ERROR 2245: Attempted to create too many ROS containers for projection *string*

ERROR 2843: Could not create thread for recoverProjectionLocal

ERROR 2844: Could not create thread for SubsessionHandler

ERROR 2845: Could not create thread for SubsessionHandler Hurry

ERROR 2997: DBDesigner memory usage (*value* bytes) exceeded system limit

ERROR 3300: Exceeded temp space cap, requested *value* with *value* remaining (used *value*) bytes

ERROR 3416: Filter tried to allocate too much memory (*value*, out of *value* allowed)

ERROR 3587: Insufficient resources to execute plan on pool *string* [*string*]

ERROR 3921: MemoryPool *string* used more memory than allowed

ERROR 3937: MIN/MAX window function could not operate in memory

ERROR 4305: Out of system WOS memory during catalog SELECT

ERROR 4764: Source tried to allocate too much memory (*value*, out of *value* allowed)

ERROR 5000: Thread limit *value*, but statement needs *value* threads

ERROR 5001: ThreadManager failed to create thread *string*: *string*

ERROR 5022: Timer service failed to run *value*: *string*

ERROR 5065: Too many ROS containers exist for the following projections: *string*

ERROR 5921: Insufficient memory available for database designer

ERROR 5924: Insufficient resources to get resource from *string* pool [*string*]

ERROR 6941: Result set size (*value* KB) is too big. Try increasing TempSpaceCap (currently *value* KB)

ERROR 7414: Error in blob creation: Allocation of chunk failed

ERROR 7423: Failed to acquire resources for blob '*string*'

ERROR 7700: Attempted to move/copy too many ROS containers for projection *string*

ERROR 7740: Tried to expand to *value* from *value*, while limit is only *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 53100

This topic lists the errors associated with the SQLSTATE 53100.

SQLSTATE 53100 Description

ERRCODE_DISK_FULL

Error messages associated with this SQLState

ERROR 2475: Cannot rebalance cluster. Insufficient disk space on the following nodes: *string*

ERROR 2927: Could not write to [*string*]: *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 53200

This topic lists the errors associated with the SQLSTATE 53200.

SQLSTATE 53200 Description

ERRCODE_OUT_OF_MEMORY

Error messages associated with this SQLState

ERROR 2296: Calloc of *value* bytes for *string* failed

ERROR 2344: Cannot allocate sufficient memory for COPY statement (*value* requested, *value* permitted)

ERROR 3499: Hash table out of memory

ERROR 3811: Join [*string*] inner partition did not fit in memory; value [*string*]

ERROR 3813: Join did not fit in memory

ERROR 3814: Join inner did not fit in memory

ERROR 3815: Join inner did not fit in memory [*string*]

ERROR 3819: Join table did not fit in memory

ERROR 3895: Malloc of *value* bytes for *string* failed

ERROR 4176: Not enough memory for test directive numTopKHeaps

ERROR 4302: Out of memory

ERROR 4303: Out of memory when expanding glob: *string*

ERROR 4357: Pattern partition will not fit into memory

ERROR 4381: Plan memory limit exhausted: [*string*]

ERROR 4495: Query *value* exceeded memory usage limit. Design result for this query might be suboptimal

ERROR 4512: Ran out of WOS memory during *string*

ERROR 4524: Realloc of *value* bytes for *string* failed

ERROR 5062: Too many hash table entries

ERROR 5063: Too many matches in a single partition

ERROR 5147: Unable to reserve memory (*value* K) for the WOS

ERROR 5952: Malloc of *value* bytes in Block Memory Manager failed

ERROR 7413: Error in blob creation: Allocation of blob memory failed

ERROR 7979: Cannot allocate all variables

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 54000

This topic lists the errors associated with the SQLSTATE 54000.

SQLSTATE 54000 Description

ERRCODE_PROGRAM_LIMIT_EXCEEDED

Error messages associated with this SQLState

ERROR 2052: *string* Row size *value* is too large

ERROR 2472: Cannot prepare statement - too many prepared statements

ERROR 3460: Function *string* may give a *value*-octet result; the limit is *value* octets

ERROR 3626: Invalid buffer enlargement request size *value*

ERROR 3866: Line is too long in timezone file "*string*", line *value*

ERROR 4282: Operator *string* may give a *value*-octet Varbinary result; the limit is *value* octets

ERROR 4283: Operator *string* may give a *value*-octet Varchar result; the limit is *value* octets

ERROR 4557: *regexp_replace* result is too long

ERROR 4913: Target lists can have at most *value* entries

ERROR 5043: Timezone directory stack overflow

ERROR 5060: Too many data partitions

ERROR 5263: Unsupported access to external table

ERROR 5265: Unsupported access to virtual schema

ERROR 5266: Unsupported access to virtual table

ERROR 5267: Unsupported access to virtual view

ERROR 5749: Array size exceeds the maximum allowed (*value*)

ERROR 6064: Transaction commit delta is too large (*value*)

ERROR 6076: Unable to fork to start spread: *value*

ERROR 6117: Size of compressed serialized plan (*value* bytes) is too large

ERROR 6147: Attempted to return result set too large; exceeds remote execution buffer size of *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 54001

This topic lists the errors associated with the SQLSTATE 54001.

SQLSTATE 54001 Description

ERRCODE_STATEMENT_TOO_COMPLEX

Error messages associated with this SQLState

ERROR 4588: Request size too big. Please try to simplify the query
ERROR 4963: The query contains a SET operation tree that is too complex to analyze
ERROR 4964: The query contains an expression that is too complex to analyze
ERROR 7913: MLA CUBE has too many columns: *value*
ERROR 7914: MLA grouping sets have *value* elements taking too much memory: *value* bytes
ERROR 7915: Too many grouping sets generated: *value*
ERROR 7916: Too many grouping sets generated: *value*
ERROR 8031: Too many grouping elements: *value*, request size too big
ERROR 8037: Query plan is too large to serialize

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 54011

This topic lists the errors associated with the SQLSTATE 54011.

SQLSTATE 54011 Description

ERRCODE_TOO_MANY_COLUMNS

Error messages associated with this SQLState

ERROR 2106: A table/projection/view can only have up to *value* columns -- this create statement has *value*

ERROR 2118: Adding column causes row size (*value*) to exceed MaxRowSize (*value*)

ERROR 2136: Aggregate function cannot have *value* input argument(s)

ERROR 2137: Aggregate function cannot have *value* return value(s)

ERROR 2181: Analytic function cannot have *value* return value(s)

ERROR 2291: Call to ColumnTypes.addAny() is not allowed in Aggregate functions

ERROR 3466: Function cannot have *value* return value(s)

ERROR 4202: Number of columns (*value*) exceeds limit (*value*)

ERROR 4481: Projection row size (*value*) exceeds MaxRowSize (*value*)

ERROR 4630: Row size exceeds MaxRowSize: *value* > *value*

ERROR 4875: Table "*string*" can only have up to *value* columns -- adding one will exceed this limit

ERROR 5898: File system cannot have *value* input argument(s)

ERROR 5899: File system cannot have *value* return value(s)

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 54023

This topic lists the errors associated with the SQLSTATE 54023.

SQLSTATE 54023 Description

ERRCODE_TOO_MANY_ARGUMENTS

Error messages associated with this SQLState

ERROR 2441: Cannot have more than *value* segmentation columns
ERROR 2469: Cannot pass more than *value* arguments to a function
ERROR 4431: Procedures cannot have more than *value* parameters
ERROR 4646: Scalar/Transform functions cannot have more than *value* parameters
ERROR 5055: Too many arguments
ERROR 5056: Too many arguments to evaluate_delete_performance function

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 55000

This topic lists the errors associated with the SQLSTATE 55000.

SQLSTATE 55000 Description

ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE

Error messages associated with this SQLState

ERROR 2088: A concurrent load into the partition or a concurrent mergeout operation interfered with this statement
ERROR 2143: AHM advanced beyond snapshot epoch
ERROR 2149: AHM can't be set
ERROR 2150: AHM can't be set while retentive refresh is running
ERROR 2151: AHM can't be set. (*value* nodes are down, out of *value*.)
ERROR 2152: AHM can't be set. (*value* nodes are down.)
ERROR 2159: All nodes must be UP to rebalance a cluster
ERROR 2163: Already released
ERROR 2175: An error occurred when loading library file on node *string*, message:
string
ERROR 2200: AnalyzeStatsPlanMaxColumns configuration parameter '*value*' invalid; must be greater than zero
ERROR 2201: AnalyzeStatsSampleBands configuration parameter '*value*' invalid; must be greater than zero

Vertica Documentation

Vertica Error Messages

ERROR 2241: Attempt to create view using an invalid relation

ERROR 2242: Attempt to run multi-node KV plan

ERROR 2294: CALL_USE_SESSION_NODES used without setting nodes

ERROR 2303: Can not tell if tables have data, too few responses (*value*) to be conclusive

ERROR 2316: Can't match imported node '*string*' to node in current database

ERROR 2371: Cannot commit DML/DDl while a node is shutting down

ERROR 2378: Cannot convert column "*string*" to type "*string*"

ERROR 2380: Cannot create a library without an initialized LibraryPath on node: *string*

ERROR 2388: Cannot create projections on a temporary table that has data

ERROR 2409: Cannot drop any more columns in *string*

ERROR 2410: Cannot drop column "*string*" since it is referenced in the default expression of column "*string*"

ERROR 2413: Cannot drop column "*string*" since it was referenced in the default expression of added column "*string*"

ERROR 2422: Cannot Drop: *string string* depends on *string string*

ERROR 2424: Cannot execute query with temporary table because a node has recovered since the start of this session

ERROR 2448: Cannot issue this command in a read-only transaction

ERROR 2459: Cannot modify temporary table *string* because a node has recovered or rebalance data took place since the start of this *string*

ERROR 2483: Cannot remove snapshots without an initialized SnapshotPath

ERROR 2496: Cannot revoke EXECUTE permission from the owner: *string*

ERROR 2497: Cannot revoke EXECUTE permission from the super user

ERROR 2505: Cannot set column "*string*" in table "*string*" to NOT NULL since it contains null values

ERROR 2512: Cannot set memoryCap for session whose current user has been dropped

ERROR 2516: Cannot set runTimeCap for session whose current user has been dropped

ERROR 2517: Cannot set tempSpaceCap for session whose current user has been dropped

ERROR 2541: Cannot use addAny() with any other input column types

ERROR 2542: Cannot use addAny() with any other output column types

ERROR 2563: Cannot validate DV storage

ERROR 2564: Cannot validate storage

ERROR 2587: Changes cannot be made to [*string*]. It has been retired

ERROR 2762: COPY: Cannot load into IDENTITY column "*string*"

ERROR 2763: COPY: Cannot specify parsing options for IDENTITY column "*string*"

ERROR 2903: Could not reset epoch because DML locks are held

ERROR 2904: Could not reset epoch because projections exist

ERROR 2933: Couldn't force partition projection *string*

ERROR 2934: Couldn't force partition projections *string*

ERROR 2954: Current phase of recovery failed due to missed event at epoch *value*

ERROR 2955: Current set of up nodes do not satisfy dependencies

ERROR 2956: Current set of up nodes do not satisfy dependencies for table *string*

ERROR 2961: Current user has been dropped so no defaults are available

ERROR 2962: Current user has been dropped so no roles are available

Vertica Documentation

Vertica Error Messages

ERROR 2969: Cursor can only scan forward

ERROR 3002: DDL statement interfered with alter column type

ERROR 3136: drop_partition failed for *string* on node *string*. The projection contains unpartitioned data

ERROR 3196: Error deserializing objects

ERROR 3229: Error loading library file:[*string*]

ERROR 3254: Error reading from file

ERROR 3278: Error writing to file

ERROR 3318: Execution aborted by node state change

ERROR 3392: Failed to update local min/max objects for column "*string*"

ERROR 3807: JobTracker::getMarkedStorages(): Unknown job *value*

ERROR 3808: JobTracker::jobComplete(*string*): Unknown job *value*

ERROR 3809: JobTracker::setDetails(*value,value,value*): Unknown job *value*

ERROR 3810: JobTracker::setJobDescription(*string*): Unknown job *value*

ERROR 3838: Key *value* already in use

ERROR 3882: Location cannot be dropped as it stores data files

ERROR 3911: maxMemorySize for *string* can be changed only when the *string* WOS is empty

ERROR 3924: merge_partitions() failed on *string* because of unpartitioned data

ERROR 3925: Mergeout failed: projection *string* is not up-to-date

ERROR 4032: Naming conflict: *string* exists

ERROR 4092: No plan received at node

ERROR 4120: No transaction running on node

ERROR 4121: No transaction running, does previous load_snapshot_prep succeeded?

ERROR 4127: No valid cache found

ERROR 4138: Node *string* is not available for queries

ERROR 4144: Node has not been set up for plan execution

ERROR 4146: Node is not active or recovering, cannot plan query

ERROR 4148: Node not prepared to accept plan

ERROR 4151: Node unprepared for rebalance

ERROR 4177: Not enough nodes are up for Projection <*string*> to be available, marking it as out of date

ERROR 4403: Portal "*string*" cannot be run

ERROR 4457: Projection *string* checkpoint epoch lags snapshot epoch

ERROR 4458: Projection *string* contains data in the WOS

ERROR 4459: Projection *string* create epoch is greater than the epoch in the query

ERROR 4462: Projection *string* has HSE > snapshot epoch and buddy *string* has HSE <= snapshot epoch

ERROR 4464: Projection *string* is not up-to-date

ERROR 4467: Projection (name: *string*, oid: *value*) is newly added during current recovery

ERROR 4485: Projections *string* contain data in the WOS

ERROR 4530: Rebalance unable to moveout all data on projection *string*

ERROR 4592: reset_epoch is disabled because the EnableResetEpoch configuration parameter is 0

ERROR 4611: Returned string value '[*string*]' with length [*value*] is greater than declared field length of [*value*] of field [*string*] at output column index [*value*]

ERROR 4698: Sequence "*string*" has been created by an IDENTITY/AUTO_INCREMENT column and cannot be dropped

Vertica Documentation

Vertica Error Messages

ERROR 4700: Sequence *string* has not been accessed in the session

ERROR 4765: Specified K-safety for projection creation is insufficient to support currently down nodes

ERROR 4791: Storage extends beyond specified segment range

ERROR 4793: Stream error: *string*

ERROR 4860: System is not k-safe. DDL is disallowed

ERROR 4861: System is not k-safe. DDL/DML is disallowed

ERROR 4879: Table "*string*" has projections in non-up-to-date state

ERROR 4880: Table "*string*" has projections that are not up-to-date that can refresh from buddy

ERROR 4903: Table or projection no longer exists

ERROR 4934: The attribute "*string*" in table "*string*" needs to be included in projection "*string*" because it is used in the partitioning expression

ERROR 4941: The data type requires length/precision specification

ERROR 4965: The restore violates K safety

ERROR 4972: The types/sizes of source column (index *value*, length *value*) and destination column (index *value*, length *value*) do not match

ERROR 5084: Tried to add field '*string*' that already exists

ERROR 5085: Tried to add unknown node '*string*' to user-defined query plan

ERROR 5132: Unable to evaluate the delete performance after dropping this column for projection "*string*"

ERROR 5151: Unable to validate data in *string*: *string*

ERROR 5204: Unknown data type

ERROR 5210: Unknown object: *string*

ERROR 5321: Usage of [*string*] cannot be changed. It has been retired

ERROR 5381: User Defined Scalar Function can only have 1 return column, but *value* is provided

ERROR 5522: A concurrent operation interfered with this statement

ERROR 5534: Can't create table in specified target schema

ERROR 5535: Can't find target table's schema

ERROR 5543: Cannot use column type 'any' with any other input column types

ERROR 5544: Cannot use column type 'any' with any other output column types

ERROR 5568: DVWos can not be moved

ERROR 5583: Fault Group "*string*" already exists in a fault group

ERROR 5587: Fault Group "*string*" not found in Fault Group "*string*"

ERROR 5590: Found *value* unsegmented projections with basename *string*; inconsistent with permanent nodes count *value*

ERROR 5626: Node "*string*" already exists in a fault group

ERROR 5627: Node "*string*" not found in Fault Group "*string*"

ERROR 5660: Source table can not be temp, virtual, system, or external

ERROR 5662: Storage tier *string* has not been found on all nodes

ERROR 5666: Table "*string*" has prejoin projections

ERROR 5667: Target table can not be temp, virtual, system, or external

ERROR 5674: TM interfered with this statement

ERROR 5676: Unable to move/swap partitions because some projection(s) contain unpartitioned data

ERROR 5705: Dvmergeout failed: projection *string* is not up-to-date

ERROR 5712: JobTracker::reportStart: Unknown job *value*

Vertica Documentation

Vertica Error Messages

ERROR 5735: Tier *string* is referenced by storage policies. Can not make storage location changes as requested

ERROR 5760: Can only change setting when all started nodes are UP

ERROR 5765: Cannot change control node away from self because other nodes depend on this node to be their control node

ERROR 5766: Cannot change final control node away from self until at least one other node is promoted to be a control node

ERROR 5772: Cannot manually alter automatically generated fault groups

ERROR 5786: Column *value* does not have corresponding storages yet. A concurrent add column operation might be running

ERROR 5883: Failed to list hcatalog tables

ERROR 6001: Recovery failed because DVROS straddles discard epoch

ERROR 6002: Recovery failed because ROS *value* [*0xvalue*, *0xvalue*] straddles endEpoch *value* to discard

ERROR 6003: Recovery failed because ROS straddles discard epoch

ERROR 6035: Table "*string*" has no non-null records under the column *key_name*

ERROR 6037: Table "*string_string*" cannot be found or was not created internally

ERROR 6065: Tried to allocate and initialize a *value*-byte string with *value* zero bytes; VString is too small

ERROR 6066: Tried to copy a *value*-byte string to *value*-byte VString object; VString is too small

ERROR 6105: View "*string*" is already linked to flex table "*string*". Linked views will not be overwritten

ERROR 6106: View "*string*" is already linked to this table. Linked views will not be overwritten

ERROR 6107: View "*string_string*" cannot be found or was not created internally

ERROR 6110: A design/deployment process is currently executing in this design space

ERROR 6121: A concurrent operation interfered with this statement

ERROR 6176: Cannot replace node *string* because it is already a STANDBY

ERROR 6177: Cannot replace node *string* because it is not DOWN

ERROR 6184: Cannot swap partition between same table

ERROR 6185: Cannot transition node *string* to *string* because it still has data

ERROR 6186: Cannot transition node *string* to STANDBY because its loss would cause the cluster to shutdown

ERROR 6207: Could not create internal data-storage directory '*string*': *value*

ERROR 6352: Need to split partitions before move/swap partitions

ERROR 6357: No standby nodes are currently available

ERROR 6359: Node *string* has not been replaced

ERROR 6360: Node *string* is a *string* node and cannot store data

ERROR 6374: Original node *string* is not currently available in STANDBY mode

ERROR 6419: Table *string* can not be temp, virtual, system, or external

ERROR 6422: Target node *string* is a *string* node, not a STANDBY node

ERROR 6424: Target standby node *string* is not currently available

ERROR 6446: UDX set BOOLEAN column *value* to non-boolean value *value*

ERROR 6478: Can only take object-level snapshot from local storage locations

ERROR 6484: Mergeout failed: ROS(es) have been dropped/moved

ERROR 6524: DDL interfered with this statement. Table is not partitioned or partition expression got changed

ERROR 6553: Compact storage failed: projection *string* is not up-to-date

Vertica Documentation

Vertica Error Messages

ERROR 6554: Compact storage failed: ROS(es) have been dropped/moved

ERROR 6633: An enabled constraint can only be declared on a global temporary table during CREATE TABLE

ERROR 6634: An enabled constraint cannot be created on a temporary table with existing data

ERROR 6646: Attempted to commit when column *value* has *value* rows while column 0 has *value* rows

ERROR 6647: Attempted to use a *string* as a *string*

ERROR 6648: Attempted to write past the end of a column

ERROR 6658: Can not *string* to the same table

ERROR 6665: Can not set priority for text index, please use its source table

ERROR 6667: Can only change setting when lockless recovery is enabled

ERROR 6674: Cannot do object level restore while a node is recovering

ERROR 6675: Cannot drop column "*string*" since it is the last non-IDENTITY, non-AUTO_INCREMENT column

ERROR 6677: Cannot execute query because table recovery status change

ERROR 6811: Incorrect use of setter in processBlock

ERROR 6812: Incorrect use of setter in processPartition for [*value*] column

ERROR 6826: Invalid node Oid *value*

ERROR 6870: Multiple values for the parameter *string*. The parameter will not be set

ERROR 6937: Restore: Cannot overwrite object *string*

ERROR 6957: Source items are not the correct encoding for direct copies

ERROR 6958: Source items are not the correct size for direct copies

ERROR 6976: Table "*string*" has not been recovered. Please try later

ERROR 6982: Table can not be temp, virtual, system, or external

ERROR 6989: Terminate() must be overridden for a User Defined Aggregate

ERROR 6997: The key *string* doesn't exist

ERROR 7001: The sessionParamReader for namespace '*string*' doesn't exist

ERROR 7002: The sessionParamWriter for namespace '*string*' doesn't exist

ERROR 7016: UDX set BOOLEAN column *value*, row *value* to non-boolean value *value*

ERROR 7017: Unable to acquire side process info

ERROR 7072: WebHCat query [*string*] failed: *string*

ERROR 7093: Comment length of parameter '*string*' is '*value*' which exceeds the maximum allowed '*value*'

ERROR 7096: Failed to rollover MinMaxObj on all nodes

ERROR 7119: Source and target table do not match: *string*

ERROR 7121: Staging table and target table do not match: *string*

ERROR 7173: Current set of up nodes do not satisfy dependencies for DFS file distribution *value*

ERROR 7193: RecoverByContainer::recover can't advance the cpe by recovering containers

ERROR 7212: Cannot restore data to node *string*

ERROR 7290: Trying to set the column "*string*" to size of *value* All data type lengths in table "*string*" must not be greater than *value*, the current maximum raw size for flex table values

ERROR 7299: Cannot alter type of column "*string*" since it is referenced in the default expression of table "*string*", column "*string*"

ERROR 7300: Cannot alter type of column "*string*" since it is referenced in the set using expression of table "*string*", column "*string*"

ERROR 7301: Cannot drop column "*string*" since it is referenced in the default expression of table "*string*", column "*string*"

ERROR 7302: Cannot drop column "*string*" since it is referenced in the set using expression of table "*string*", column "*string*"

ERROR 7343: *string* expression of IDENTITY/AUTO_INCREMENT column "*string*" cannot be altered

ERROR 7386: Cannot proceed with the revoke. User *string* would be left without access to a resource pool

ERROR 7390: Cannot set idlesessiontimeout for session whose current user has been dropped

ERROR 7448: Location cannot be dropped as it stores DFS files

ERROR 7458: Mergeout failed: found missed *string* on table *string*

ERROR 7537: Sequence "*string*" has been created by an IDENTITY/AUTO_INCREMENT column and cannot be used in a *string* expression

ERROR 7583: User or Role *value* cannot be found

ERROR 7617: SBJobTracker::jobComplete: Unknown projOid "*value*"

ERROR 7618: SBJobTracker::jobComplete: Unknown txnid "*value*"

ERROR 7619: SBJobTracker::setAdditionalInfo: Unknown oid "*value*"

ERROR 7671: Wrong checksum for library file *string*

ERROR 7766: Cannot drop column "*string*" since it is referenced in the set using expression of column "*string*"

ERROR 7811: SendFiles on node *string*: file [*string*] has changed

ERROR 7821: Cannot rename user "*string*" since they use MD5 password format

ERROR 7856: Not all requested nodes are available at this moment

ERROR 7857: Not enough available nodes at this moment

ERROR 7866: DDL statement interfered with this statement. ProjCol *value* cannot be found

ERROR 7978: Can only use KV hint with SELECT queries

ERROR 7983: Cannot use KV Hint with a subquery

ERROR 8009: Running KV query inside existing transaction. There will be a performance implication. Consider rerunning the query outside of an existing transaction

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 55006

This topic lists the errors associated with the SQLSTATE 55006.

SQLSTATE 55006 Description

ERRCODE_OBJECT_IN_USE

Error messages associated with this SQLState

ERROR 2060: *string* WOS is not empty; cannot renew. Do a moveout

ERROR 2307: Can't drop self

ERROR 3003: DDL statement interfered with Database Designer

ERROR 3004: DDL statement interfered with query replan

ERROR 3896: Manual mergeout not supported while tuple mover is running

ERROR 3897: Manual moveout not supported while tuple mover is running

ERROR 4122: No up-to-date super projection left on the anchor table of projection *string*

ERROR 4139: Node *string* transitioned to state UP during this statement

ERROR 4145: Node is active and cannot be altered

ERROR 4455: Projection *string* cannot be dropped because K-safety would be violated

ERROR 4470: Projection cannot be dropped because history after AHM would be lost

ERROR 4488: Projections cannot be dropped or data would be lost due to down nodes

ERROR 4527: Rebalance is already running

ERROR 4528: Rebalance is already scheduled to run in the background

ERROR 4882: Table "*string*" is used as a dimension in a prejoined projection

ERROR 4896: Table (*value*) has been dropped

ERROR 4971: The status of one or more nodes changed during query planning

ERROR 6052: The system must retain at least one control node after the drop

ERROR 6162: Cannot alter a control node to be a STANDBY node

ERROR 6163: Cannot alter initiator node to STANDBY

ERROR 7219: A DDL statement interfered with this statement: constraint '*string*' has been disabled or dropped on table '*string*'

ERROR 7220: A DDL statement interfered with this statement: constraint '*string*' has been enabled on table '*string*'

ERROR 7402: DDL statement interfered with this statement. Text indices don't line up

ERROR 7616: SBJobTracker::createMarker: existing proj0id *value*

ERROR 7888: DDL statement (*string*) interfered with this statement

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 55V02

This topic lists the errors associated with the SQLSTATE 55V02.

SQLSTATE 55V02 Description

ERRCODE_CANT_CHANGE_RUNTIME_PARAM

Error messages associated with this SQLState

ERROR 4324: Parameter *string* will not take effect until database restart

ERROR 7567: This Parameter has been disabled and will not be set

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 55V03

This topic lists the errors associated with the SQLSTATE 55V03.

SQLSTATE 55V03 Description

ERRCODE_LOCK_NOT_AVAILABLE

Error messages associated with this SQLState

ERROR 5156: Unavailable: *string* - Locking failure: *string*

ERROR 5157: Unavailable: [Txn value] *string* - *string* error *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 55V04

This topic lists the errors associated with the SQLSTATE 55V04.

SQLSTATE 55V04 Description

ERRCODE_TM_MARKER_NOT_AVAILABLE

Error messages associated with this SQLState

ERROR 2082: A *string* operation is already in progress on projection *string.string* [container *value* txnid *value* session *string*]

ERROR 2083: A *string* operation is already in progress on projection *string.string* [txnid *value* session *string*]

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 57014

This topic lists the errors associated with the SQLSTATE 57014.

SQLSTATE 57014 Description

ERRCODE_QUERY_CANCELED

Error messages associated with this SQLState

ERROR 2246: Audit canceled

ERROR 2279: Bulk Import canceled

Vertica Documentation

Vertica Error Messages

ERROR 2310: Can't find projection

ERROR 2325: Canceled (in *string*)

ERROR 2326: Canceled: *string* - Locking canceled: *string*

ERROR 2327: Canceled: [Txn *value*] *string* - *string string*

ERROR 2576: Catchup recovery interrupted

ERROR 2704: Connection canceled

ERROR 2996: DBDesigner canceled by user

ERROR 3286: evaluate_delete_performance canceled

ERROR 3319: Execution canceled (compile)

ERROR 3320: Execution canceled (prepare)

ERROR 3321: Execution canceled (start)

ERROR 3322: Execution canceled by operator

ERROR 3323: Execution got unlucky!

ERROR 3324: Execution intentionally failed

ERROR 3326: Execution time exceeded run time cap of *string*

ERROR 3515: import_catalog_objects canceled

ERROR 4114: No super projection available for analyze_statistics

ERROR 4142: Node failure during execution

ERROR 4143: Node failure in *string*

ERROR 4287: Operator intervention on *string*

ERROR 4380: Plan canceled prior to execute call

ERROR 4439: Processing aborted by peer on *string*

ERROR 4496: Query canceled while waiting for resources

ERROR 4787: Statement abandoned due to subsequent DDL

ERROR 4789: Statement is canceled

ERROR 4843: Subsession interrupted

ERROR 5757: build_flextable_view canceled

ERROR 5787: compute_flextable_keys canceled

ERROR 5915: Hcatalog webservice query canceled

ERROR 5953: materialize_flextable_columns canceled

ERROR 6292: Internal query raised exception during ALTER TABLE ADD CONSTRAINT

ERROR 6389: Rebalance task *string* canceled

ERROR 6535: Table Owner lock canceled

ERROR 7088: Can not perform the upgrade because 'EnableMinMaxScaling' configuration parameter is turned off. Please turn it on first

ERROR 7115: Receive on *string*: query has been canceled

ERROR 7278: StopExecution BeforePlan

ERROR 7279: StopExecution CompilePlan

ERROR 7280: StopExecution ExecutePlan

ERROR 7281: StopExecution InitPlan

ERROR 7282: StopExecution Plan

ERROR 7283: StopExecution PreparePlan

ERROR 7284: StopExecution SerializePlan

ERROR 7457: Mergeout canceled

ERROR 7462: Moveout canceled

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 57015

This topic lists the errors associated with the SQLSTATE 57015.

SQLSTATE 57015 Description

ERRCODE_SLOW_DELETE

Error messages associated with this SQLState

ERROR 5822: Detected slow delete

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 57V01

This topic lists the errors associated with the SQLSTATE 57V01.

SQLSTATE 57V01 Description

ERRCODE_ADMIN_SHUTDOWN

Error messages associated with this SQLState

ERROR 3556: Initiating node is down

ERROR 4150: Node status is not UP

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 57V03

This topic lists the errors associated with the SQLSTATE 57V03.

SQLSTATE 57V03 Description

ERRCODE_CANNOT_CONNECT_NOW

Error messages associated with this SQLState

ERROR 2863: Could not fork UDX zygote process, *string*

ERROR 2929: Couldn't create new UDX side process, failed to get UDX side process info from zygote: *string*

ERROR 2930: Couldn't create new UDX side process, the language *string* is not supported

ERROR 2937: Couldn't set TCP_NODELAY option, might get latency in RPC message delivery: *string*

ERROR 3363: Failed to connect to side process, *string*

ERROR 3364: Failed to connect to UDX zygote, *string*

ERROR 3366: Failed to create new UDX side process, couldn't connect to it: *string*

ERROR 4720: Session manager cannot add an external session - disabled

ERROR 5699: Cannot find java binary: neither the Linux environment variable JAVA_HOME nor Vertica config parameter JavaBinaryForUDx is set

ERROR 5702: Couldn't create new UDX side process: *string*

ERROR 5803: Couldn't create new UDX side process, failed to set locale information: *string*

ERROR 6707: Could not fork UDX zygote process: *string*

ERROR 6712: Couldn't cancel the external procedure: *string*

ERROR 6715: Couldn't execute the external procedure: *string*

ERROR 6840: Key generation for the zygote failed: [*string*]

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 57V04

This topic lists the errors associated with the SQLSTATE 57V04.

SQLSTATE 57V04 Description

ERRCODE_DML_COMMIT_DURING_SHUTDOWN

Error messages associated with this SQLState

ERROR 6523: Cannot commit DML while a node is shutting down

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 58030

This topic lists the errors associated with the SQLSTATE 58030.

SQLSTATE 58030 Description

ERRCODE_IO_ERROR

Error messages associated with this SQLState

ERROR 2024: *string* Error occurred during BZIP decompression. BZIP error code: *value*

ERROR 2026: *string* Error occurred during ZLIB decompression. ZLIB error code: *value*, Message: *string*

Vertica Documentation

Vertica Error Messages

ERROR 2253: Bad return from WaitForMultipleObjects: *value* (*value*)

ERROR 2432: Cannot get LibraryPath from node: *string*

ERROR 2433: Cannot get MD5 checksum from node: *string*

ERROR 2600: Checksums do not match (computed=*0xvalue*, fromdisk=*0xvalue*) discarding checkpoint!

ERROR 2674: ColumnAccessBase open error

ERROR 3197: Error deserializing snapshot info from file *string*

ERROR 3255: Error reading from file *string*

ERROR 3303: Exception during measurement deserialization

ERROR 3305: Exception during Stats deserialization:*string*

ERROR 3370: Failed to create socket waiting event: *value*

ERROR 3385: Failed to reset socket waiting event: *value*

ERROR 3408: File size on disk does not match catalog for *string*

ERROR 3412: FileColumnReader: Get block *string* @ *value* error

ERROR 3478: getnameinfo_all() failed: *string*

ERROR 3550: Info file *string* does not exist

ERROR 3796: IO_ERROR writing data file [*string*]

ERROR 4364: Performance measurement of [*string*] failed

ERROR 4377: Pixw finish error

ERROR 4378: Pixw open error

ERROR 4379: Pixw write error

ERROR 4518: Read error when expanding glob: *string*

ERROR 4632: RowAccessBase open error

ERROR 5124: Unable to close catalog file [*string*]

ERROR 5126: Unable to create catalog file [*string*]

ERROR 5131: Unable to drop catalog file [*string*]

ERROR 5133: Unable to fsync catalog file [*string*] errno=*value*

ERROR 5141: Unable to open file [*string*]

ERROR 5152: Unable to write catalog file [*string*]

ERROR 5153: Unable to write checksum to catalog file [*string*]

ERROR 5154: Unable to write object to catalog file [*string*]

ERROR 5887: Failed to mount file system *value*: *string*

ERROR 5901: Filesystem does not pass basic test: *string*

ERROR 5902: Filesystem does not pass basic test: I/O data differ

ERROR 6074: Unable to close catalog file after fsync [*string*] errno=*value*

ERROR 6077: Unable to fsync catalog dir [*string*] errno=*value*

ERROR 6079: Unable to open catalog dir fd for fsync [*string*] errno=*value*

ERROR 6080: Unable to open catalog dir for fsync [*string*] errno=*value*

ERROR 6081: Unable to open catalog file for fsync [*string*] errno=*value*

ERROR 6082: Unable to open spread conf file *string* for writing

ERROR 6084: Unable to stat file *string*: *string*

Vertica Documentation

Vertica Error Messages

ERROR 6118: *string* Error occurred during LZ0 decompression (compressed data violation).LZ0 error code:
value

ERROR 6258: Exception during file writer deserialization: *string*

ERROR 6259: Exception during file writer serialization: *string*

ERROR 6260: Exception during snapshot deserialization: *string*

ERROR 6261: Exception during snapshot serialization: *string*

ERROR 6262: Exception during storage container deserialization: *string*

ERROR 6263: Exception during storage container serialization: *string*

ERROR 6527: Filesystem does not support snapshot

ERROR 6528: Filesystem failed to restore snapshot

ERROR 6550: Cannot open catalog source file *string*

ERROR 6746: Duplicate storage location id: *value*

ERROR 6762: Error manifest format

ERROR 6776: Failed to glob [*string*] because of error: *string*

ERROR 6808: Improperly ordered or duplicate storage ids: *string*, *string*

ERROR 6830: Invalid section for storage locations

ERROR 6860: Malformed object line: *string*

ERROR 6861: Malformed storage location line: *string*

ERROR 6874: No Library tar file:*string*

ERROR 6984: tar_append_file: *string*: *value*

ERROR 6985: tar_append_tree failed:*value* Real dir:*string*; save dir: *string*

ERROR 6986: tar_close failed:*value*

ERROR 6987: tar_extract_all failed:*value* Extract path:*string*

ERROR 6988: tar_open failed:*value* Path:*string*

ERROR 7178: Error loading from all sources

ERROR 7718: Exception on closing file: *string*

ERROR 7720: Exception on flushing file: *string*

ERROR 7721: Exception on opening file: *string*

ERROR 7722: Exception on reading file: *string*

ERROR 7723: Exception on resizing file: *string*

ERROR 7724: Exception on writing to file: *string*

ERROR 7728: No place to store chunk files

ERROR 7782: Cannot create Blobs directory

ERROR 7869: No files match when expanding glob: [*string*]

ERROR 7881: Unable to parse aws auth

ERROR 7882: Unable to parse azure auth

ERROR 7883: Unable to read AWS_ACCESS_KEY_ID from env

ERROR 7884: Unable to read AWS_REGION from env

ERROR 7885: Unable to read AWS_SECRET_ACCESS_KEY from env

ERROR 7886: Unable to read AZURE_ACCESS_KEY_ID from env

ERROR 7887: Unable to read AZURE_SECRET_ACCESS_KEY from env

ERROR 8025: Exception when open blob file: *string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE 58V01

This topic lists the errors associated with the SQLSTATE 58V01.

SQLSTATE 58V01 Description

ERRCODE_UNDEFINED_FILE

Error messages associated with this SQLState

ERROR 3664: Invalid filename. Input filename is an empty string

ERROR 6222: Depends can specify files only. [*string*] is not a valid file

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE V1001

This topic lists the errors associated with the SQLSTATE V1001.

SQLSTATE V1001 Description

ERRCODE_LOST_CONNECTIVITY

Error messages associated with this SQLState

ERROR 2709: Connection to spread closed

ERROR 4048: NetworkReceive: Decompression failed
ERROR 4054: NetworkSend on *string*: failed to open connection to node *string* (*string*)
ERROR 4140: Node *string* was not successfully added to the cluster
ERROR 4533: Receive: Decompression failed
ERROR 4534: Receive on *string*: Message receipt from *string* failed [*string*]
ERROR 4541: ReceiveFiles on *string*: Unexpected end of stream from *string* [*string*]
ERROR 4547: RecvFiles on *string*: Open failed on node [*string*] (*string*)
ERROR 4572: RemoteSend: Open failed on node [*string*], IPAddr is [*string*], port is [*value*] (*string*)
ERROR 4683: Send: Connection not open [*string* tag:*value* plan *value*]
ERROR 4684: Send: Open failed on node [*string*] (*string*)
ERROR 4689: SendFiles on *string*: Open failed on node [*string*] (*string*)
ERROR 5579: Failure in send on socket *string*: *string*
ERROR 5624: NetworkReceive on *string*: failed to open connection to node *string* (*string*)
ERROR 5625: NetworkReceive on *string*: Message receipt from *string* failed: *string*
ERROR 5658: Send on *string*: Open failed on node [*string*] (Address lookup for *string*(*string*) failed)
ERROR 7116: Receive on *string*: open failed for node *string* (*string*) --- has query been cancelled?

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE V1002

This topic lists the errors associated with the SQLSTATE V1002.

SQLSTATE V1002 Description

ERRCODE_K_SAFETY_VIOLATION

Error messages associated with this SQLState

ERROR 2406: Cannot drop *value* nodes from a *value* node cluster with *value* nodes down - cluster would appear partitioned and database would shutdown. Bring some nodes up and try again
ERROR 2529: Cannot support K=*value* on only *value* nodes
ERROR 2957: Current system KSAFE level is not fault tolerant
ERROR 4477: Projection KSAFE *value* can not be met with only *value* nodes
ERROR 4478: Projection KSAFE override *value* cannot be less than current system K-safe value *value*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE V1003

This topic lists the errors associated with the SQLSTATE V1003.

SQLSTATE V1003 Description

ERRCODE_CLUSTER_CHANGE

Error messages associated with this SQLState

ERROR 2094: A node has come UP which missed ALTER COLUMN check
ERROR 2095: A node has come UP which missed drop partition keys check
ERROR 2096: A node has come UP which missed partitioning check
ERROR 2097: A node has entered the cluster since the session started
ERROR 2098: A node has entered the cluster since the session was started
ERROR 2099: A node has entered/left the database cluster
ERROR 3428: Following nodes are UP but not in the backup node set: *string*
ERROR 5312: Up node set changed during restore
ERROR 5523: A node has come UP which missed ADD COLUMN statement
ERROR 6650: Backup node *string* mapped to restore node *string*, but restore node is not in the cluster
ERROR 6876: No nodes up!
ERROR 7214: Node types changed during restore

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE V2000

This topic lists the errors associated with the SQLSTATE V2000.

SQLSTATE V2000 Description

ERRCODE_AUTH_FAILED

Error messages associated with this SQLState

ERROR 3493: GSS error: *string*. Error details: (*string/string*)

ERROR 3718: Invalid old password

ERROR 6635: An error occurred during GSS authentication

ERROR 6636: An error occurred during kerberos authentication

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE V2001

This topic lists the errors associated with the SQLSTATE V2001.

SQLSTATE V2001 Description

ERRCODE_LICENSE_ISSUE

Error messages associated with this SQLState

ERROR 2382: Cannot create another node. The current license permits *value* node(s) and the database catalog already contains *value* node(s)

ERROR 3248: Error parsing license end date

ERROR 3863: License issue: *string*

ERROR 4943: The Enterprise Edition is installed. You cannot downgrade from the Enterprise Edition to the Community Edition

ERROR 5943: License corrupt: *string* requires license *string*, but it is corrupt

ERROR 5944: License expired: *string* requires license *string*, but it has expired

ERROR 5946: License issue: *string* (required by *string*)

ERROR 5955: Missing license: *string* requires license *string*

ERROR 6164: Cannot alter STANDBY node. The current license permits *value* node(s) and the database catalog already contains *value* node(s)

ERROR 6549: Cannot install new license to the database. New license permits *value* node(s) but the database catalog already contains *value* node(s)

ERROR 6753: Error *string*

ERROR 7018: Unable to audit license: database lacks an active license!

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VC001

This topic lists the errors associated with the SQLSTATE VC001.

SQLSTATE VC001 Description

ERRCODE_CONFIG_FILE_ERROR

Error messages associated with this SQLState

ERROR 3833: Kerberos keytab file must be owned by the database user, and have no permissions for "group" or "other"

ERROR 4951: The Kerberos keytab file is either empty or too small in size to be valid

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VD001

This topic lists the errors associated with the SQLSTATE VD001.

SQLSTATE VD001 Description

ERRCODE_DESIGNER_FUNCTION_ERROR

Error messages associated with this SQLState

- ERROR 2010: *string* cannot be NULL
- ERROR 2012: *string* clause does not exist in the query
- ERROR 2202: Anchor table for projection *string* does not exist, so it cannot be added to deployment
- ERROR 2204: Anchor table of projection *string* is a Session scoped table
- ERROR 2205: Anchor table of projection *string* is a System table
- ERROR 2211: API *string* not available in old DBD engine
- ERROR 2212: API cannot take query input file and query string, only one can be set
- ERROR 2304: Can only load *value string* under the *string* design type
- ERROR 2328: Cannot *string* as design was created already
- ERROR 2336: Cannot add another Comprehensive design to deployment *string*
- ERROR 2337: Cannot add design projections in extend catalog type deployment *string* in workspace *string*
- ERROR 2338: Cannot add design tables to design *string* because there are populated designs
- ERROR 2339: Cannot add design to deployment *string* because design *string* has not been populated
- ERROR 2369: Cannot clear design tables from design *string* because there are populated designs
- ERROR 2375: Cannot compute projections to be dropped for only incremental designs deployment
- ERROR 2394: Cannot design encoding for Projection *string* as it does not have any AUTO encoded columns
- ERROR 2395: Cannot design encoding for Projection *string* as it is not SAFE -- Create its buddies
- ERROR 2396: Cannot design/deploy for virtual system schema *string*
- ERROR 2423: Cannot execute deployment when there are non-up-to-date safe projections for table *string*
- ERROR 2454: Cannot load invalid query: *string*
- ERROR 2456: Cannot load queries as design was populated already
- ERROR 2463: Cannot output design projections because design is not available
- ERROR 2464: Cannot output query because query id is invalid
- ERROR 2471: Cannot populate drop projections in extend catalog type deployment *string* in workspace *string*
- ERROR 2477: Cannot refresh projections for table *value* as it was dropped
- ERROR 2480: Cannot remove any design table from design *string* because there are populated designs
- ERROR 2485: Cannot remove workspace *string* because it does not exist
- ERROR 2492: Cannot retrieve design tables for design *string* in workspace *string*
- ERROR 2493: Cannot retrieve information for design *string* in workspace *string*
- ERROR 2507: Cannot set k-safety when design *string* has been populated
- ERROR 2657: Column '*string*' does not exist in Table *string*
- ERROR 2658: Column '*string*' is duplicated in the column list
- ERROR 3053: Deployment *string* already exists in workspace *string*

Vertica Documentation

Vertica Error Messages

ERROR 3054: Deployment got canceled

ERROR 3056: Deployment ksafety should be equal or greater than design ksafety. Deployment ksafety is *value* and design ksafety is *value*

ERROR 3057: Deployment name cannot be NULL

ERROR 3058: Deployment Projections status is set to Error

ERROR 3060: Design *string* already exists

ERROR 3061: Design *string* already exists for workspace *string*

ERROR 3063: Design *string* has already been added to deployment *string*

ERROR 3064: Design *string* has not been populated in workspace *string* so projection cannot be added

ERROR 3065: Design *string* hasn't been populated

ERROR 3066: Design *string* in workspace *string* is not available

ERROR 3067: Design *string* is already populated

ERROR 3068: Design *string* is populated, remove design first (designer_remove_design)

ERROR 3071: Design name cannot have more than *value* characters

ERROR 3072: Design name may contain only alphanumeric or underscore characters

ERROR 3073: Design did not complete successfully, so deployment did not start

ERROR 3074: Design K-safety should be 0

ERROR 3077: Design name cannot have character '.'

ERROR 3079: Optimization objective cannot be NULL

ERROR 3080: Design query with design_query_id *value* does not exist

ERROR 3081: Design Query with design_query_id *string* does not exist

ERROR 3082: Design *string* does not exist

ERROR 3087: design_override_type *string* for query (design_query_id *value*) already exists

ERROR 3088: design_override_type *string* for table *string* already exists

ERROR 3089: design_override_type *string* for table *string* does not exist

ERROR 3100: Did not find any projections to design encodings for

ERROR 3101: Did not find design projections for projection ids given

ERROR 3102: Did not find design projections for tablePattern *string*

ERROR 3103: Did not find design tables to add

ERROR 3104: Did not find design tables to remove

ERROR 3105: Did not find projection id *value* in deployment *string* in workspace *string*

ERROR 3106: Did not find projections for design *string* in workspace *string*

ERROR 3107: Did not find rows in deployment table for deployment *string* in workspace *string*

ERROR 3108: Did not find rows in designs table for workspace *string*

ERROR 3140: Dropping design without getting design projections, API call is of no use

ERROR 3166: Empty design name is not allowed

ERROR 3188: Error after projection refresh: *string*

ERROR 3194: Error creating workspace: Invalid workspace name

ERROR 3195: Error deleting deployment status table

ERROR 3202: Error during deployment while setting ksafety before deployment starts

ERROR 3203: Error during design: *string*

ERROR 3205: Error during drop projections: *string*

Vertica Documentation

Vertica Error Messages

ERROR 3208: Error during projection creation: *string*

ERROR 3214: Error during remove design *string*

ERROR 3215: Error during rename projections: *string*

ERROR 3241: Error opening query input file [*string*]

ERROR 3250: Error querying deployment projections statements table

ERROR 3251: Error querying deployment projections table

ERROR 3252: Error querying design projections table for design *string* in workspace *string*

ERROR 3253: Error querying: *string*

ERROR 3266: Error status for projections to add for table *string*

ERROR 3267: Error status for projections to drop for table *string*

ERROR 3268: Error updating deployment projections table

ERROR 3270: Error while loading statistics into design tables for design *string*

ERROR 3277: Error writing to [*string*]

ERROR 3356: External table *string* is not a design table

ERROR 3358: Failed during select mark_design_ksafe(*value*)

ERROR 3415: Filename cannot be NULL

ERROR 3480: Given design *string* does not exist

ERROR 3489: Group-by override *value* on query *value* cannot be satisfied

ERROR 3543: Incremental design needs a query or an input query file to be set

ERROR 3574: INSERT query without SELECT is not supported: *string*

ERROR 3649: Invalid Deploy Operation string *string*

ERROR 3650: Invalid deploy status string *string*

ERROR 3740: Invalid query input file [*string*]

ERROR 3795: Invalid design creator name

ERROR 3817: Join override *value* on query *value* cannot be satisfied

ERROR 3824: K cannot be *value* (maximum allowed is *value*)

ERROR 3825: K must be equal to or greater than *value*, cannot reduce current k-safety level

ERROR 3827: K-safety cannot be NULL

ERROR 3867: List of projections cannot be NULL

ERROR 3898: mark_design_ksafe(*value*) failed; some projections may not be k-safe

ERROR 4031: Namespace for LOCAL temporary tables cannot be used to add design tables

ERROR 4057: New ksafety cannot be less than 0

ERROR 4078: No deployment data in *string.string*

ERROR 4080: No drop entries found for deployment *string* in workspace *string*

ERROR 4099: No projections found for the projection ids string *string*

ERROR 4103: No rows in deployment projections table

ERROR 4117: No tables found in schema *string*

ERROR 4118: No tables found in the table pattern given

ERROR 4119: No tables to design projections for

ERROR 4235: One of the design tables no longer exist

Vertica Documentation

Vertica Error Messages

ERROR 4311: Override (*design_override_id value*) is ignored because the table *string* is no longer a design table

ERROR 4312: Override (*design_override_id value*) is ignored because the table does not exist

ERROR 4313: *override_type string* for query (*design_query_id value*) does not exist

ERROR 4314: *override_type string* is invalid

ERROR 4460: Projection *string* does not exist

ERROR 4461: Projection *string* does not exist

ERROR 4466: Projection *string* to be refreshed was dropped

ERROR 4475: Projection id cannot be NULL

ERROR 4476: Projection id list cannot be NULL

ERROR 4479: Projection name cannot be NULL

ERROR 4497: Query Id cannot be NULL

ERROR 4498: Query referencing EPOCH column is not supported

ERROR 4499: Query referencing local temporary table *string* is not supported: *string*

ERROR 4500: Query referencing projection *string* is not supported: *string*

ERROR 4501: Query without referencing any catalog table is not supported: *string*

ERROR 4503: Query table *string* does not exist

ERROR 4504: Query table contains multiple entries with *qid = value*

ERROR 4505: Query weight must be a positive number

ERROR 4525: Rebalance data cannot proceed when there are non-up-to-date projections in the catalog

ERROR 4526: Rebalance data failed during select mark_design_ksafe(*value*)

ERROR 4651: Schema *string* does not exist

ERROR 4652: Schema *string* is not a designer created schema, so it cannot be dropped

ERROR 4655: Schema name cannot be NULL

ERROR 4721: Session scoped table *string* is not a design table

ERROR 4783: Start deploy: deploy is already running on this node

ERROR 4819: Subqueries in UPDATE/DELETE is not supported: *string*

ERROR 4866: System table *string* is not a design table

ERROR 4874: Systems tables within system schema *string* cannot be added as design tables

ERROR 4885: Table *string* does not exist

ERROR 4886: Table *string* does not exist anymore in the catalog

ERROR 4888: Table *string* has no statistics or data. As a result, the proposed projections on this table may be suboptimal

ERROR 4890: Table *string* is not a design table

ERROR 4891: Table *string* is not a design table, referenced in query (*qid=value*): *string*

ERROR 4902: Table name cannot be NULL

ERROR 4907: Table pattern cannot be NULL

ERROR 4920: Terminated after SO enum. See log for the content of the SOs

ERROR 4942: The design table entry with table name *string.string* is corrupted, as that table has been renamed in the Vertica catalog

ERROR 4976: There are *value* nodes. Deployment K = *value* is not possible

Vertica Documentation

Vertica Error Messages

ERROR 4977: There are no projections to add in deployment *string* for workspace *string* so no projections can be dropped

ERROR 4980: There is 1 node. Deployment K = *value* is not possible

ERROR 4981: There is more than one design *string* in workspace *string*

ERROR 4983: There is no design tables system table in workspace *string*

ERROR 4991: This invalid query cannot be loaded: *string*

ERROR 4994: This non-SELECT query is not supported: *string*

ERROR 4995: This query is not supported in DBDesigner

ERROR 5363: User *string* does not have privileges to access design table: *string*

ERROR 5364: User *string* does not have privileges to access table: *string*

ERROR 5390: User has insufficient privileges on table *string*

ERROR 5480: Workspace *string* cannot be a virtual system schema

ERROR 5481: Workspace *string* does not exist

ERROR 5482: Design *string* is configured for extend_catalog so no designs can be computed

ERROR 5483: Design *string* is configured for extend_catalog so remove drops is not supported

ERROR 5484: Design *string* is configured for extend_catalog so there are no design tables

ERROR 5485: Design *string* is configured for extend_catalog, there are no design tables

ERROR 5486: Workspace cannot be NULL

ERROR 5487: Design name cannot be NULL

ERROR 5564: Deployment Parallelism cannot be less than zero

ERROR 5565: Deployment parallelism cannot be NULL

ERROR 5573: Error generating results set

ERROR 5575: Error querying designs table

ERROR 5588: Fenced mode false is not supported for *string* functions

ERROR 5589: Fenced mode is not supported for SQL functions

ERROR 5591: Hurryup parameter cannot be NULL

ERROR 5597: Invalid input query: '*string*'

ERROR 5650: Query without referencing any design tables is not supported: *string*

ERROR 5657: Segmentation type of the projection *string* is not supported for encoding design, skipping

ERROR 5694: Weight for query_text '*string*' is '*value*'. Only positive weight values are accepted

ERROR 5747: analyzeStats flag cannot be NULL

ERROR 5773: Cannot output deployment script because design is not available

ERROR 5792: continueAfterError flag cannot be NULL

ERROR 5817: Deploy flag cannot be NULL

ERROR 5855: Did not find any tables to analyze correlations on

ERROR 5857: dropDesignAndCtx flag cannot be NULL

ERROR 5858: dropProjs flag cannot be NULL

ERROR 5866: Error while analyzing correlations for design table *string.string*

ERROR 5867: Error while analyzing count distincts for design table *string.string*

ERROR 5868: Error while analyzing count distincts on correlation sample for design table *string.string*

ERROR 5869: Error while analyzing segmentation skew for design table *string.string*

ERROR 5870: Error while dropping existing correlations in design table *string*

ERROR 5871: Error while loading or analyzing correlations in design tables for design *string*

ERROR 5907: Force option cannot be NULL

ERROR 5908: forceRecomputation flag cannot be NULL

ERROR 5919: Input cannot be NULL

ERROR 5938: isAdminUser flag cannot be NULL

ERROR 5939: K-safety of incremental designs must match the current system k-safety (which is *value*)

ERROR 5979: onlyScript flag cannot be NULL

ERROR 5980: outputScript flag cannot be NULL

ERROR 6040: Table *string* has no correlations

ERROR 6041: Table *string* has no data. As a result, no correlations were analyzed on this table

ERROR 6042: Table *string* has no statistics or data. As a result, no correlations were analyzed on this table

ERROR 6043: Table *string* has no statistics or data. As a result, no correlations were read into this table

ERROR 6049: The mode of analyzing correlations cannot be NULL

ERROR 6050: The mode of analyzing correlations is invalid

ERROR 6096: User has insufficient privileges on table *string.string*

ERROR 6223: Design K-safety should be in [*0,value*] range

ERROR 6247: Error during deployment: no projections found for deployment *string*

ERROR 6489: Error querying v_catalog.projections table

ERROR 7191: Projection with expressions *string* is not supported for encoding design

ERROR 7192: Projection with expressions *string* is not supported for encoding design, skipping

ERROR 7379: Cannot generate unique name for Database Designer schema

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VP000

This topic lists the errors associated with the SQLSTATE VP000.

SQLSTATE VP000 Description

ERRCODE_USER_PROC_ERROR

Error messages associated with this SQLState

ERROR 2059: *string* with specified name and parameters does not exist: *string*

ERROR 2315: Can't have more than one parameters with the same name: *string*

ERROR 3354: External procedures directory has not been set

ERROR 3355: External procedures have not been installed

ERROR 3465: Function cannot be moved into a system schema

ERROR 4322: Parameter must have names

ERROR 4323: Parameter type is not valid for an external procedure: *string*

ERROR 4373: Phase *value* of multi-phase transform function marked prepass

ERROR 4430: Procedures cannot be created in a system schema

ERROR 5232: Unrecognized identifier: *string*

ERROR 5368: User Defined Aggregates do not support fenced execution mode

ERROR 5372: User Defined Function type not found

ERROR 5374: User Defined Scalar Function *string* is giving bad numeric precision *value*, the maximum is *value*

ERROR 5375: User Defined Scalar Function *string* is giving bad string typmod *value*, the minimum is *value*

ERROR 5376: User Defined Scalar Function *string* is giving typmod of precision *value*, larger than the max precision *value*

ERROR 5377: User Defined Scalar Function *string* provided non-zero precision (*value*) for Interval Year To Month

ERROR 5378: User Defined Scalar Function *string* provided precision *value*, larger than the maximum precision *value*

ERROR 5379: User Defined Scalar Function *string* provided range for Day To Second, but the function's return type is Interval Year To Month

ERROR 5380: User Defined Scalar Function *string* provided range for Year To Month, but the function's return type is Interval Day To Second

ERROR 5684: User Defined Extension cannot be created in a system schema

ERROR 6051: The schema has been dropped

ERROR 6576: Schema does not exist: *string*

ERROR 6580: Stemmer Udx *string* must have VARCHAR or LONG VARCHAR parameter type

ERROR 6581: Stemmer Udx *string* must have VARCHAR or LONG VARCHAR return type

ERROR 6642: Argument types must be specified for stemmer "*string*"

ERROR 6869: Multi-phase transform function must have atleast one phase

ERROR 6961: Stemmer Udx *string* must return a single argument

ERROR 7010: Tokenizer required to create text index

ERROR 7015: UDSF or SQL Function with specified name and parameters does not exist: *string*

ERROR 7082: "*string*" is an incorrect EarlyMaterialize directive

ERROR 7128: Tokenizer "*string*" with specified argument types could not be found

ERROR 7129: Tokenizer Udx *string* must be polymorphic or have a single input field of CHAR, VARCHAR, LONG VARCHAR, VARBINARY, LONG VARBINARY, or USER DEFINED argument type

ERROR 7139: View cannot be moved into a system schema

ERROR 7582: User Defined Transforms with cursors do not support fenced execution mode

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VP001

This topic lists the errors associated with the SQLSTATE VP001.

SQLSTATE VP001 Description

ERRCODE_USER_PROC_EXEC_ERROR

Error messages associated with this SQLState

ERROR 2376: Cannot connect to UDX side process (pid = *value*) during cancel: *string*

ERROR 2837: Could not create pipe for user procedure execution, errno=*value*

ERROR 2853: Could not execute user procedure: fork error

ERROR 2858: Could not find function definition

ERROR 2861: Could not find running procedure for *string*, proc ID=*value*

ERROR 3223: Error in calling *string*() for User Defined Function *string* at [*string:value*], error code: *value*, message: *string*

ERROR 3224: Error in calling *string*() for User Defined Scalar Function *string* at [*string:value*], error code: *value*, message: *string*

ERROR 3398: Failure in UDX RPC call *string*() (pid = *value*): *string*

ERROR 3399: Failure in UDX RPC call *string*(): *string*

ERROR 4424: Procedure execution error: exit status=*value*

ERROR 4425: Procedure execution error: procedure killed by signal (*value*)

ERROR 4538: Received message with unexpected type *string*

ERROR 5170: Unexpected exception from in calling *string*() for User Defined Scalar Function *string*

ERROR 5171: Unexpected exception in calling *string*() in User Defined Function *string*

ERROR 5205: Unknown error killing procedure *string*

ERROR 5395: User procedure execution failed

ERROR 5398: User-defined Analytic Function *string* produced fewer output rows than input rows

ERROR 5399: User-defined Scalar Function *string* outputted a timezone (*value*) not in allowed range (*value*, *value*)

Vertica Documentation

Vertica Error Messages

ERROR 5400: User-defined Scalar Function *string* produced fewer output rows (*value*) than input rows (*value*)

ERROR 5430: Vertica process is not allowed to kill procedure *string*

ERROR 5580: Failure sending parameters block because the *value* parameters require *value* bytes, which exceeds the maximum size of *value* bytes

ERROR 5604: Invalid procedure file: [*string*]

ERROR 5638: Procedure file [*string*] cannot be owned by root

ERROR 5639: Procedure file [*string*] must be executable by vertica user (dbAdmin)

ERROR 5640: Procedure file [*string*] must be owned by specified procedure user

ERROR 5641: Procedure file [*string*] must enable set UID attribute

ERROR 5656: Root cannot execute external procedure

ERROR 5683: User '*string*' not found on node

ERROR 5861: Error calling *string*() in User Function *string* at [*string:value*], error code: *value*, message: *string*

ERROR 5863: Error during setting up function, message: *string*

ERROR 6085: Unexpected exception calling *string*() User Function in *string*

ERROR 6086: Unexpected exception calling *destroyUDxFenced*()

ERROR 6087: Unexpected exception thrown by *UDFileSystem* at [*string:value*], error code: *value*, message: *string*

ERROR 6668: Can't access [*string*]: No filesystem is mapped to scheme *string*

ERROR 6706: Could not find filesystem for scheme *string*

ERROR 6713: Couldn't cancel the user procedure, *string*

ERROR 6756: Error in calling destructor for *UDFilter* function at [*string:value*], error code: *value*, message: *string*

ERROR 6757: Error in calling destructor for *UDParser* function at [*string:value*], error code: *value*, message: *string*

ERROR 6758: Error in calling destructor for *UDSource* function at [*string:value*], error code: *value*, message: *string*

ERROR 6759: Error in calling *~string*() for User Defined Function *string* at [*string:value*], error code: *value*, message: *string*

ERROR 6760: Error in calling *~string*() for User Defined Scalar Function *string* at [*string:value*], error code: *value*, message: *string*

ERROR 6783: Filesystem does not support glob *string*

ERROR 6859: *makeConnection: send_msg(value)* did not succeed: *string*

ERROR 7025: Unexpected exception in calling destructor in *UDFilter* function

ERROR 7026: Unexpected exception in calling destructor in *UDParser* function

ERROR 7027: Unexpected exception in calling destructor in *UDSource* function

ERROR 7028: Unexpected exception in calling *~string*() for User Defined Scalar Function *string*

ERROR 7029: Unexpected exception in calling *~string*() in User Defined Function *string*

ERROR 7097: Failure in *UDx RPC call string()*

ERROR 7112: Procedure reported: *string*

ERROR 7685: User-defined Scalar Function *string* produced more output rows (*value*) than input rows (*value*)

ERROR 7712: Error during cleanup for User Defined Function *string: string*

ERROR 7981: Cannot reserve memory from the JVM resource pool

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VX001

This topic lists the errors associated with the SQLSTATE VX001.

SQLSTATE VX001 Description

ERRCODE_INTERNAL_ERROR

Error messages associated with this SQLState

ERROR 2025: *string* Error occurred during BZIP initialization. BZIP error code: *value*

ERROR 2027: *string* Error occurred during ZLIB initialization. ZLIB error code: *value*, Message: *string*

ERROR 2111: Active queue cleared while running

ERROR 2405: Cannot do boundary analysis on type *value*

ERROR 2616: Cluster recovery failed, try again

ERROR 2928: Couldn't check this session's state

ERROR 3099: Did not find a variable

ERROR 3198: Error dropping table partition, data in WOS

ERROR 3211: Error during recovery running *string* queries, cannot continue: *string* (at *string:value*)

ERROR 3212: Error during recovery running *string*: *string* (at *string:value*)

ERROR 3220: Error generating query for: *string*

ERROR 3245: Error parsing *string*

ERROR 3257: Error retrieving *string* in *string*: *string*

ERROR 3292: Event apply failed

ERROR 3302: Exception decoding the response we just locally encoded

ERROR 3483: Got unexpected error code from spread: *value*, *string*

ERROR 3818: JOIN qualifications to not refer to the correct relation(s)

ERROR 3969: More than one variable found

ERROR 4372: *pg_analyze_and_rewrite* for View query failed

ERROR 4514: Raw parse of View query string failed

ERROR 4545: Recovery Error: Cannot get projections on local node

ERROR 5236: Unrecognized node type *value*

Vertica Documentation

Vertica Error Messages

ERROR 5237: Unrecognized node type *value* in postprocess conditions

ERROR 5526: Already have a ready_recv *string*, ignoring

ERROR 5539: Cannot find buddy projection's statistics for collecting row counts, min and max

ERROR 5540: Cannot find buddy projections for collecting row counts, min and max

ERROR 5541: Cannot find the up-nodes of buddy projection for collecting row counts, min and max

ERROR 5679: Unrecognized order by expression

ERROR 5680: Unrecognized select column list

ERROR 5695: With query is not a Select Statement

ERROR 5719: Path Sampling failed. Try a different random seed for the pathSampling hint

ERROR 5802: Could not stop all dirty transactions[txnId = *string*]

ERROR 5865: Error while analyzing approximate count distincts on table *string.string*

ERROR 6062: Too Many User defined types

ERROR 6248: Error occurred during LZ0 decompression: LZ0 checksum error on compressed data, possibly due to file corruption

ERROR 6249: Error occurred during LZ0 decompression: LZ0 checksum error on uncompressed data, possibly due to file corruption

ERROR 6250: Error occurred during LZ0 decompression: LZ0 expected destination length larger than BLOCK_SIZE, possibly need to recompile lzop, or set --blocksize to a larger value

ERROR 6251: Error occurred during LZ0 decompression: LZ0 expected destination length larger than MAX_BLOCK_SIZE, possibly due to file corruption

ERROR 6252: Error occurred during LZ0 decompression: LZ0 expected source length is wrong, possibly due to file corruption

ERROR 6253: Error occurred during LZ0 header processing: expecting more than *value* bytes, possibly file corrupted

ERROR 6254: Error occurred during LZ0 header processing: return code *value*, possibly due to file corruption

ERROR 6428: The password for "*string*", encryption algorithm *string* does not match the effective server configured encryption algorithm *string*, please expire the password to reset

ERROR 6429: The sending password for "*string*", encryption algorithm *string* does not match the effective server configured encryption algorithm *string*

ERROR 6440: Trying to change password for "*string*", but password encryption algorithm does not match, server configured *string*, client send in *string*

ERROR 6468: Wrong Password Security Algorithm *string*

ERROR 6482: Failed to parse Access Policies for table "*string*" [*string*]

ERROR 6678: Cannot extract relations in the query

ERROR 6738: DML is running while collecting dirty txns

ERROR 6768: Error retrieving Group ROS [*value*] of ROS [*value*]

ERROR 6771: Fail to get table recovery status when node is not INITIALIZING/RECOVERING/READY/UP

ERROR 6775: Failed to generate an annotated query: *string*

ERROR 6779: Failed to recover all tables, would retry!

ERROR 6780: Failed to recover node, shutting down...

ERROR 6820: Input query cannot be deparsed

ERROR 6821: Input query is not supported

ERROR 6893: Output annotated query is not supported

ERROR 6895: Output stream failed to initialize

ERROR 6927: Query not ready to write to export file

ERROR 6930: Recovery Error: Cannot get projections of table *string*

ERROR 6931: Recovery Error: Cannot get projections of tables having prejoin projections

ERROR 6942: ROS [*value*] is in a bundle without a storageId

ERROR 6995: The content of the input query saved previously changed

ERROR 7032: Unexpected segmentation for constraint projection

ERROR 7274: Recovery Error: projection recovery start epoch is behind AHM in *string* phase. Has AHM been advanced during recovery?

ERROR 7348: *value* projections out of *value* projections fail to moveout to epoch *value*

ERROR 7493: OpenSSL RAND_bytes returns error; *value*

ERROR 7535: ROS *value* starts at epoch *value*, end at epoch *value*, straddle truncate epoch *value*

ERROR 7645: Configured password type *string* not admissable under FIPS

ERROR 7656: Effective password type *string* not admissable under FIPS

ERROR 7661: MD5 not permitted in FIPS mode

ERROR 7668: The server password *string* is not allowed on FIPS systems; please have DB admin correct this

ERROR 7669: Trying to change password for "*string*", but MD5 hash algorithm not permitted

ERROR 7855: Found SAL corruption

ERROR 7873: Property *string* is in *string*, not *string*

ERROR 7874: Property *string* not found in *string*

ERROR 7944: Cannot set SET-USING because refresh_columns failed

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VX002

This topic lists the errors associated with the SQLSTATE VX002.

SQLSTATE VX002 Description

ERRCODE_DATA_CORRUPTED

Error messages associated with this SQLState

ERROR 2940: CRC Check Failure Details:
File Name: *string*
File Offset: *value*
Compressed size in file: *value*
Memory Address of Read Buffer: *value*
Pointer to Compressed Data: *value*
Memory Contents:
string

ERROR 3030: Delete: could not find a data row to delete (data integrity violation?)

ERROR 3218: Error finalizing ROS DataTarget

ERROR 3219: Error flushing data file [*string*]

ERROR 3409: FileColumnReader: block *string* @ *value* 's CRC *value* doesn't match record *value*

ERROR 3410: FileColumnReader: Decompression error in *string* at offset *value*

ERROR 4762: Sort Order Violation:
Row Position: *value*
Column Index: *value*
Last Row: *string*
This Row: *string*

ERROR 5704: Delete (Join): could not find a data row to delete (data integrity violation?)

ERROR 6767: Error reading from orc parser input stream [*string*]: file shorter than expected, read to *value*, requested to *value*

ERROR 7581: Unsupported model_type

ERROR 7767: Error reading from parquet parser input stream [*string*]: file shorter than expected, read to *value*, requested to *value*

ERROR 7859: Block[*value*] has unknown type. (size: *value*, type: *value*, pad: *value*, count: *value*, position: *value*)

ERROR 7860: Block[*value*]->count is not consistent with other blocks. (*value* != *value*)

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VX003

This topic lists the errors associated with the SQLSTATE VX003.

SQLSTATE VX003 Description

ERRCODE_INDEX_CORRUPTED

Error messages associated with this SQLState

ERROR 3544: Index corruption. *string: string*

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Error Messages Associated with SQLSTATE VX004

This topic lists the errors associated with the SQLSTATE VX004.

SQLSTATE VX004 Description

ERRCODE_PLAN_TO_SQL_INTERNAL_EROR

Error messages associated with this SQLState

ERROR 6890: Optimizer-generated annotated query has unexpected error. Please report to Vertica

Note: For information about troubleshooting Vertica error messages, see the [Vertica Knowledge Base](#) and the [Vertica Forum](#).

Glossary

The Vertica Glossary defines terms that are common and specific to Vertica.

Access Rank

Determines the speed at which a column can be accessed. Columns are stored on disk from the highest ranking to the lowest ranking in which the highest ranking columns are placed on the fastest disks and the lowest ranking columns are placed on the slowest disks.

ACID

ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably, as follows:

- Atomicity — All of the tasks of a transaction are performed or none of them are. This means that if the entire transaction succeeds, it is committed. If any part of the transaction fails, the entire transaction fails and is rolled back.
- Consistency — Data integrity constraints are maintained. This ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not). Vertica provides limited constraint checking.
- Isolation — Concurrent transactions do not interact with one another.
- Durability — Short of catastrophic failure, the effects of committed transactions persist.

Administration Host

The host on which the Vertica rpm package was manually installed. Always run the Administration Tools on this host if possible.

Administration Tools

One of the ways you can administer an Vertica database is provided in the form of a graphical user interface, called Administration Tools, which lets you perform various tasks quickly and easily. You can also use Administration Tools to connect to a database using vsql.

Always run the Administration Tools on the Administration Host if possible.

```
$ /opt/vertica/bin/adminTools
```

Note: Throughout the Micro Focus documentation, you might see Administration Tools referred to as Admin Tools or admintools or adminTools. They all refer to the same utility.

Agent

A daemon process that runs on each Vertica cluster node. The agent is used by certain clients, such as Management Console, to administer Vertica.

Agents monitor Vertica database clusters and communicate with their clients to provide the following functionality:

- Provide local access, command, and control over database instances on a given node, using functionality similar to Administration Tools
- Report log-level data from the Administration Tools and Vertica log files
- Cache details from long-running jobs—such as create/start/stop database operations—that you can view through your browser
- Track changes to data-collection and monitoring utilities and communicate updates to clients
- Specifically for MC, communicate between all cluster nodes and MC through a webhook subscription, which automates information sharing and reports on cluster-specific issues like node state, alerts, events, and so on

The agent runs on port 5444, which must be accessible to agent clients.

Aggregate Projection

See [live aggregate projection](#).

AHM

An abbreviation for Ancient History Mark, AHM is the epoch prior to which historical data can be purged from physical storage.

Anchor Table

Database table that is the source for data in a [projection](#) .

Ancient History Mark

Is the epoch prior to which historical data can be purged from physical storage. Abbreviated throughout the documentation as AHM.

Apportioned Load

An *apportioned load* is a divisible load, such that you can load a single data file on more than one node. Each load starts at a different offset, requiring a parser that supports appportioning. You must use a User-Defined Load (UDL) parser that supports appportioning. An example is provided in the SDK (`DelimFilePortionParser`).

ATM

See Automatic Tuple Mover (ATM).

Authentication

The process of verifying the identity of a user attempting to connect to a database.

Authentication Service (AS)

A service that usually on the same host as the Kerberos Key Distribution Center (KDC), the AS issues tickets for a desired service, which are in turn given to users for access to the service. The AS answers requests from clients that do not send credentials with a request. The AS is generally used to gain access to the ticket-granting service (TGS) service by issuing a ticket-granting ticket (TGT).

Authorization

The process of verifying that a user has permission to perform a certain operation, such as query a specific table.

Automatic Tuple Mover

The Automatic Tuple Mover (ATM) refers to the Tuple Mover operations that occur automatically at regular intervals set by configuration parameters. For example, two configuration parameters control the TM mergeout and moveout intervals, which are both ATM activities. Conversely, invoking the `do_tm_task` function is not an ATM activity. For information about changing the TM configuration parameters, see [Tuple Mover Parameters](#) in the Administrator's Guide for further information.

Base Name

Naming convention for projections and buddy projections. The base name is the name of the associated table, followed by characters that Vertica appends to the table name. All buddy projections have the same base name so that they can be identified as a group.

Blade Server

Consists of some number of small servers, called blades, in a common chassis that allows blades to share common components such as power supplies, cooling fans, and switching capabilities.

Bit

An abbreviation for **binary digit** and is the smallest unit of data in a computer. Bits have a single binary value, 1 or 0, and these settings can be implemented as logical values (true or false) or as a switch (on or off).

Computers generally store data and execute instructions in bit multiples called bytes. There are eight bits in a byte.

Note: In some systems, the term octet is used for an eight-bit unit instead of byte.

Bitstring

A sequence of bits.

Byte

A basic unit of measurement of information storage in computers. In most systems a byte is made up of of eight bits. Half a byte (four bits) is called a nibble.

Buddy Projection

Required for K-safety. Two projections are considered to be buddies if they contain the same columns and have the same hash segmentation, using different node ordering.

Bulk Loading

A process of loading large amounts of data, such as an initial load of historic data.

C-Store

A research project at MIT, Brandeis, Brown, and the University of Massachusetts (Boston), on which Vertica is based.

Cardinality

Refers to the number of unique values for a given column in a relational table:

- *High cardinality* — Refers to columns containing values that are highly unique, such as a customer ID or an employee e-mail address. For example, in the Vertica [VMart schema](#), the `employee_dimension` table contains an `employee_key` column. This column contains values that uniquely identify each employee. Since the values in this column are unique and could be numerous, the column's cardinality type is referred to as high cardinality.
- *Normal cardinality* — Refers to columns containing values that are less unique, such as job titles and street addresses. An example of a normal-cardinality column would be `job_title` or `employee_first_name` in the `employee_dimension` table, where many employees could share the same job title or same first name.
- *Low cardinality* — Refers to a low number of unique values, relative to the overall number of records in a table. For example, in the `employee_dimension` table, the column called `employee_gender` would contain two unique values: 'Male' or 'Female'. Since there are only two values possible in this column, cardinality is low.

Case-Folded

Uppercase is converted to lowercase or lowercase is converted to uppercase.

Catalog

A set of files that contains information (metadata) about the objects in a database (nodes, tables, constraints, projections, etc.) Vertica maintains a catalog on each node in the cluster.

Catalog Path

A storage location used to store the database catalog.

Checkpoint

Every time the Tuple Mover performs a move-out operation for a given projection, it records a Checkpoint Epoch for that projection, representing an epoch up to which the projection has no data in the WOS. The minimum checkpoint epoch across all projections on that node is called the node's **Checkpoint Epoch**. It represents a point in time up to which all the data was moved out to disk. If a K-safe=1 database experiences a single-node failure, the node's recovery process attempts to rebuild the data beyond the Checkpoint Epochs from other nodes. If all nodes fail, such as during a power outage, Vertica recovers the database back to the minimum Checkpoint Epochs across all the nodes, known as the Last Good Epoch (LGE).

Cluster

The concept of Cluster in the Vertica Analytics Platform is a collection of hosts with the Vertica software packages (RPM or DEB) that are in one admin tools domain. You can access and manage a cluster from one admintools initiator host.

Columnar Tables

Vertica database tables consisting of structured data columns. The term differentiates these tables from Flex (or Flexible) tables, which minimally contain one column of unstructured, or semi-structured data. Flex tables can also have structured data columns, but they are not required. Compare with [Flexible Tables](#).

Comprehensive Design

Using the Database Designer, a comprehensive design creates a complete initial or replacement design. If you have reasonably representative data, you can create a comprehensive design that is optimized for storage and load. If you provide queries, the design is optimized for those queries.

Compression

The process of transforming data into a compact format. Compressed data cannot be directly processed; it must first be decompressed. Vertica uses integer packing for unencoded integers and LZ0 for compressible data. Although compression is generally considered to be a form of encoding, the terms have different meanings in Vertica.

Connection

A SQL connection is an occurrence of an interactive user or an application program requesting and being granted access to a database through a network connection. In Vertica, the scope of a connection is the same as that of a session.

Control node

In a large-cluster arrangement, Vertica Analytics Platform delegates control message responsibilities to a subset of nodes, called control nodes, to improve control message performance. Control nodes communicate with each other, and non-control nodes are assigned to a control node for control message communications. See [Large Cluster](#) in the Administrator's Guide.

Correlated columns

Two columns are correlated if the value of one column is related to the value of the other column. For example, state name and country name columns are strongly correlated because the city name usually, but perhaps not always, identifies the state name. The city of Conshohocken is uniquely associated with Pennsylvania, whereas the city of Boston exists in Georgia, Indiana, Kentucky, New York, Virginia, and Massachusetts. In this case, city name is strongly correlated with state name.

Cost

A relative metric of which query plan is better, based on data distribution [statistics](#). Cost does not translate into time but is an internal estimate that the query optimizer uses to compare different plans. Costs are not relative to the overall estimated run time of queries, rather they are an estimate of the overall resources that the query plan will use. If the cost of one plan is twice as large as the cost of a second plan, that does not mean the second plan will take twice as long to run.

Critical Node

A critical node is a node whose failure would cause the database to become unsafe and force a shutdown. Nodes can become critical for the following reasons:

- A node has the only copy of a particular projection.
- Fewer than half of your nodes are active.

The [V_MONITOR.CRITICAL_NODES](#) system table lists the critical nodes, if any, in your cluster.

CSV

Short for comma-separated values, another name for the comma-delimited format of data representation in tabular format. For example: Smith,Joe,123 Main St,555-4615.

Current epoch

The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written.

Data Collector

A utility that collects and retains database monitoring information.

Note: Data Collector works in conjunction with an advisor tool called Workload Analyzer, which intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history.

Data Collector retains history of important system activities and records essential performance and resource utilization counters. You can use information the Data Collector retains in the following ways:

- As a reference for what actions users have taken
- To locate performance bottlenecks
- To identify potential improvements to Vertica configuration

Data Collector is on by default. If you need to change this behavior, see [Data Collector Parameters](#) in the Administrator's Guide. See also the SQL Reference Manual for information about [data collection control functions](#).

Data Path

A storage location that contains actual database data files.

Data Warehouse

A relational database that is designed for query and analysis rather than transaction processing. Data warehouses:

- Are often subjected to a heavy load of periodic and ad hoc queries.
- Contain historical information that enables analysis of correlations and trends over long periods of time.
- Integrate data from various production (transactional) databases. Extraction, transformation, and loading (ETL) software converts the data to a common format and copies it into a data warehouse at regular intervals.
- Typically consist of one or more star or snowflake schemas.

Database

A cluster of nodes that, when active, can perform distributed data storage and SQL statement execution through administrative, interactive, and programmatic user interfaces.

Database administrator

The Linux user account that owns the database catalog and data storage on disk. The database administrator (DBA) can bypass all database authorization rules.

Database Designer

A tool that analyzes a logical schema definition, sample queries, and sample data, and creates a physical schema (projections) in the form of a SQL script that you deploy automatically or manually. The script creates a minimal set of superprojections to ensure K-safety, and, optionally, non-superprojections. In most cases, the projections created by the Database Designer provide excellent query performance within physical constraints. You can also [create custom designs](#) if the Database Designer does not meet your needs.

Database superuser

The automatically-created database user who has the same name as the Linux database administrator account and who can bypass all GRANT/REVOKE authorization, or any user that has been granted the PSEUDOSUPERUSER role. Do not confuse the concept of a database superuser with Linux superuser (root) privilege. A database superuser cannot have Linux superuser privilege.

DBA

See [Database administrator](#).

DDL

Data Definition Language commands, such as CREATE, ALTER, and DROP, let you define, alter, and drop logical schema objects. For example, you can create a table or schema or user account, alter a user or table column, and grant and revoke privileges and roles. After you create a table, you use the DML command INSERT to insert data into the table.

DELTAVAL (delta encoding)

Stores only the differences between sequential data values instead of the values themselves.

Derived Column

A column whose values are calculated by an expression at load time. The expression is specified within the COPY statement, and the column exists in the target database.

Design

Specific to the Database Designer, a design is made up of parameters (K-safety, optimization objective, and design type), projections, and queries. The design contains the design queries for which you optimize the design, query information tables and the design output tables.

Deterministic

Deterministic functions return the same result every time they are called if given the same input values.

Dialog

A Linux utility that creates user interfaces to shell scripts or other scripting languages, such as perl. In Vertica 8.1., Dialog is used to implement the Administration Tools, which runs in a terminal window.

Dimension Table

Sometimes called a lookup or reference table, a dimension table is one of a set of companion tables to a large (fact/anchor) table in a star schema. It contains the PRIMARY KEY column corresponding to the join columns in fact tables. For example, a business might use a dimension table to contain item codes and descriptions.

Dimension tables can be connected to other dimension tables to form a hierarchy of dimensions in a snowflake schema.

Directed Query

A saved set of instructions to the optimizer how to generate a query plan for a given query, in the form of SQL annotated with hints. A directed query pairs two components:

- *Input query*: A query that triggers use of this directed query when it is active.
- *Annotated query*: A SQL statement with embedded optimizer [hints](#), which instruct the optimizer how to create a query plan for the specified input query.

Dirty Read

Occurs when one transaction reads uncommitted changes made by another concurrent transaction. Vertica prevents dirty reads from occurring.

DML

Data Manipulation Language statements query or manipulate data in existing logical schema objects. Common commands are SELECT, INSERT, UPDATE, DELETE, and MERGE. For example, you can use DML statements to:

- View (SELECT) data in one or more tables or views
- Change (UPDATE) column values in existing table rows
- View (EXPLAIN) the optimizer execution plan
- Add (INSERT) new rows to a table
- Remove (DELETE) rows from a table
- Insert (MERGE) a batch of new records while simultaneously updating existing

While DDL statements let you change the *structure* of the database, DML statement lets you query or change the *contents*.

Dynamic SQL

A programmatic interface to a database management system that lets SQL statements be defined and run at run time, usually based on user input.

EOF

Is a condition in a computer operating system where no more data can be read from a data source. It refers to end-of-file. The data source is usually called a file or stream.

Source: Wikipedia

Encoding

The process of converting data into a standard format. In Vertica, encoded data can be processed directly, while compressed data cannot. Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order.

Epoch

A logical unit of time in which a single change is made to data in the system.

Epoch Map

A catalog object that provides mapping between time and epochs. Specifically, an epoch map contains a list of epoch numbers and their associated timestamps.

Equality Predicate

A predicate in a query that uses one of the following relational operators: $<$, $>$, $<=$, $>=$, $=$, $<>$, $<=>$.

ETL

(Extract, Transform, Load) is a process in data warehousing that involves extracting data from outside sources, transforming it to fit a specific schema, and ultimately loading it into the database.

Executor Node

Any node that participates in executing a specific SQL statement. The initiator node can, and usually does, also function as an executor node.

Expression

The components of a query (columns, constants, functions) that compare one or more values against other values. Expressions can perform calculations when they are combined using arithmetic and/or logical operators and are generally used in a conditional statement.

External Procedure

A procedure external to Vertica that you create, maintain, and store on the server.

Event Series

Tables with a time column, most typically a timestamp data type.

Fact Table

The table that represents quantitative or factual data. For example, in a business, a fact table might represent orders.

A fact table is often located at the center of a star schema or snowflake schema and might also be referred to as the anchor table. It typically has a large number of records and is surrounded by a collection of dimension tables, each with fewer records. The fact table participates in a join with every dimension table. It can contain data but generally contains many join columns (with optional FOREIGN KEY constraints), each of which corresponds to the primary key column of a dimension table.

FIFO

An acronym for First In/First Out, FIFO is an abstraction for organizing and manipulating data relative to time and prioritization. The expression describes the principle of a queue processing technique or servicing conflicting demands by ordering process by first-come, first-served (FCFS) behavior: what comes in first is handled first, what comes in next waits until the first is finished, and so on.

Source: Wikipedia

Flexible Tables

Vertica database tables that minimally contain two columns:

`__identity__` – A real column with an incrementing IDENTITY value for partitioning and sorting purposes, if no other columns serve this purpose.

`__raw__` – A real LONG VARBINARY column containing unstructured, or semi-structured data.

You can create Flex tables with additional real columns, but they are not required. Compare with [Columnar Tables](#).

Foreign Key

A column that is used to join a table to other tables to ensure referential integrity of the data. A FOREIGN KEY constraint is a rule that states that a column can contain only values from the PRIMARY KEY column on a specific table.

Flattening (subqueries and views)

Occurs when a subquery or named view is internally rewritten so the subquery is combined with the outer query block. The result sets of the original and flattened queries are exactly the same, but the flattened query usually benefits from significant performance improvements.

Full Backup

Consists of copying each catalog and all data files (ROS containers) on each node, as well as the complete `/opt/vertica/config` directory.

GENERAL Pool

A special built-in pool that represents the total amount of RAM available to the resource manager for use by queries. Other pools can borrow memory from the GENERAL pool. See also [Built-In Pools](#).

Grant

Vertica defines GRANT in two ways:

1. Grant a user privileges to access database objects using any [GRANT statement](#), except GRANT (Authentication):
2. Associate a user-defined authentication method with a user through [GRANT \(Authentication\)](#). This operation differs from GRANT <privileges> as authentication methods are “associated” with a user or role and privileges are “granted” to a user or role.

Examples

Grant Access Privileges to a User

This example shows how to grant a user, Joe, privileges to access the `online_sales` schema:

```
=> GRANT USAGE ON SCHEMA online_sales TO Joe;
```

Associate an Authentication Method with a User

This example shows how to associate the `v_ldap` authentication method to user `jsmith`:

```
=> GRANT AUTHENTICATION v_ldap TO jsmith;
```

Grid Computing

A software environment based on open standards and protocols that make it possible to share disparate, loosely-coupled resources across organizations and geographies. Resources can potentially include almost any component: computer cycles, storage spaces, databases, applications, files, sensors, or scientific instruments.

Grouped ROS

A grouped ROS is a highly-optimized, read-oriented physical storage structure organized by projection. A grouped ROS makes heavy use of compression and indexing. Unlike a ROS, a grouped ROS stores data for two or more grouped columns in one disk file.

Hash Function

A reproducible method of converting data into a number that can serve as a digital fingerprint of the data. Hash functions take up to 32 arguments, usually column names, and select a specific node for each row, based on the values of the columns for that row. Use one of the built-in hash functions to segment projections. These functions provide even distribution of data over a set of nodes in a cluster.

Hash segmentation

Specifies how to segment projection data for distribution across some or all cluster nodes. You can specify segmentation for a table and a projection. If a table definition specifies segmentation, Vertica uses it for that table's [auto-projections](#).

It is strongly recommended that you use Vertica's built-in [HASH](#) function, which distributes data evenly across the cluster, and facilitates optimal query execution.

Heuristics

Denotes a rule of thumb for solving problems without extensive application of an algorithm.

Hexadecimal

A numeral system with a base of 16. Hexadecimal uses sixteen symbols: 0–9 (representing values zero to nine) and A-F (representing values ten to fifteen). For example:

```
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 10 ...
```

Each hexadecimal digit represents four binary digits (bits)—also called a nibble.

To convert the base-16 hexadecimal value 0xA0FF to its base-10 decimal equivalent, hexadecimal progresses from right to left using the following places that represent a power of 16, starting with 16^0 :

```
4096  256  16  1 A    0  F  F
```

Given that A=10 and F=15 in hex, the decimal result is calculated as follows:

```
(10 x 4096) + (0 x 256) + (15 x 16) + (15 x 1) = 41215
```

Source: Wikipedia

Histogram

Provides a compact summary of large tables to permit efficient query planning. Histograms store information about how the column's values are distributed within its range.

Historical Data

Refers to any data in memory or physical storage other than the current epoch. Historical data includes all COPY, INSERT, UPDATE, and DELETE operations, including deleted rows. This allows Vertica to run a historical query on data written up to and including the epoch representing the specified date and time.

Historical query

Vertica can run a query from a snapshot of the database taken at a specific date and time or at a specific epoch. The syntax is:

```
AT TIME 'timestamp' SELECT...  
AT EPOCH epoch_number SELECT...  
AT EPOCH LATEST SELECT...
```

The command queries all data in the database up to and including the specified epoch or the epoch representing the specified date and time, without holding a lock or blocking write operations. The specified `TIMESTAMP` and `epoch_number` values must be greater than or equal to the Ancient History Mark epoch.

Historical queries, also known as snapshot queries, are useful because they access data in past epochs only. Historical queries do not need to hold table locks or block write operations because they do not return the absolute latest data. Their content is private to the transaction and valid only for the length of the transaction.

Historical queries behave in the same manner regardless of transaction isolation level. Historical queries observe only committed data, even excluding updates made by the current transaction, unless those updates are to a temporary table..

Be aware that there is only one snapshot of the logical schema. This means that any changes you make to the schema are reflected across all epochs. If, for example, you add a new column to a table and you specify a default value for the column, all historical epochs display the new column and its default value.

Host

A computer system with a 32-bit (non-production use only) or 64-bit Intel or AMD processor, RAM, hard disk, and TCP/IP network interface (IP address and hostname). Hosts share neither disk space nor main memory with each other.

ICU

An abbreviation for International Components for Unicode, ICU is an open source library that provides Unicode support.

ISO Week-Year

According to the ISO-8601 standard, the week starts on Monday, and the first week of a year contains January 4. Thus, an early January date can sometimes be in the week 52 or 53 of the previous calendar year. For example:

```
=> SELECT YEAR_ISO('01-01-2016'::DATE), WEEK_ISO('01-01-2016'), DAYOFWEEK_ISO('01-01-2016');
YEAR_ISO | WEEK_ISO | DAYOFWEEK_ISO
-----+-----+-----
      2015 |        53 |             5
(1 row)
```

Immutable (invariant) functions

When run with a given set of arguments, immutable functions such as [AVG\(\)](#) always produce the same result, regardless of environment or session settings such as locale. Some immutable functions can take an optional stable argument; in this case they are treated as stable functions.

Incremental Backup

Backs up only those files that have been added to the database since the last backup was performed. Vertica supports incremental backups through file management utilities. Incremental backups are possible because Vertica never modifies data files. Vertica adds files to and deletes them from the database. Once you have created a full backup with vbr and any subsequent incremental backups, you can use vbr to do a full restore. There is no incremental restore facility.

Incremental Design

Using the Database Designer, an incremental design creates a design with additional projections optimized for one or more queries that you provide to the design process.

Initiator Node

In the context of a client connection, the initiator node is the node associated with the specific host to which the connection was made. The initiator node can, and usually does, also function as an executor node, and is generally where the most descriptive log messages reside.

Instance

An instance of Vertica consists of the running Vertica process and disk storage (catalog and data) on a host. Only one instance of Vertica can be running on a host at any time.

Interpolation

The process of constructing new data points within a range of known data points.

JDBC

An abbreviation for Java Database Connectivity, JDBC is a call-level application programming interface (API) that provides connectivity between Java programs and data sources (SQL databases and other non-relational data sources, such as spreadsheets or flat files). JDBC is included in the Java 2 Standard and Enterprise editions.

K-Safety

K-safety sets fault tolerance for the database cluster, where K can be set to 0, 1, or 2. The value of K specifies how many copies Vertica creates of [segmented projection](#) data. If K-safety for a database is set to 1 or 2, Vertica creates K+1 instances, or *buddies*, of each projection segment. Vertica distributes these buddies across the database cluster, such that projection data is protected in the event of node failure. If any node fails, the database can continue to process queries so long as buddies of data on the failed node remain available elsewhere on the cluster.

Last Epoch

The last epoch is the current epoch minus one. `SELECT` statements under `READ COMMITTED` isolation read from the last epoch.

Last Good Epoch

A term used in manual recovery, LGE (Last Good Epoch) refers to the most recent epoch that can be recovered.

LGE

Abbreviation for last good epoch.

Live Aggregate Projection

A live aggregate projection contains columns with values that are aggregated from columns in its anchor table. When you load data into the table, Vertica aggregates the data before loading it into the live aggregate projection. On subsequent loads—for example, through [INSERT](#) or [COPY](#)—Vertica recalculates aggregations with the new data and updates the projection.

Locale

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate `GROUP BY` and `ORDER BY` clauses, joins, and the analytic `ORDER BY` clause.

The default locale for a Vertica database is `en_US@collation=binary` (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default `en_US@collation=binary` collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to `en_US_POSIX`. You can set the locale back to the default locale and collation by issuing the `vsq` meta-command `\locale`. For example:

```
=> set locale to '';
INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation=binary;
```

You can set locale through [ODBC](#), [JDBC](#), and [ADO.net](#).

Logical Schema

Consists of a set of tables and referential integrity constraints in an Vertica database. The objects in the logical schema are visible to SQL users. The logical schema does not include projections, which make up the physical schema.

Location Label

A label assigned to a storage location. Location labels identify the location so you can create object storage policies. The labeled location you use in a storage policy becomes the default storage location for the object.

LZO

LZO (Lempel-Ziv-Oberhumer) is a data-compression algorithm that focuses on decompression speed. The algorithm is lossless, with a thread-safe reference implementation. LZO is one of several supported input formats for bulk-loading data.

Man-In-The-Middle Attack

A network security breach in which a third party makes independent connections with a client and server and relays messages between them, intercepting data throughout the process. Vertica supports the SSL and Kerberos protocols to avoid this type of attack.

Management Console

A database management tool that provides a unified view of your Vertica database and lets you monitor multiple clusters from a single point of access. See [Management Console](#) in Vertica Concepts.

MC Super (superuser administrator)

Called Super on the MC interface, the MC super is the Linux user account that gets created when you [configure MC](#). See [Configuring MC](#) and [SUPER Role \(mc\)](#).

MC-managed Database

An Vertica database that an MC SUPER or ADMIN user creates on MC or imports into the MC interface, along with the database cluster. When MC users are granted database privileges, their privileges are defined though the MC itself and pertain only to databases managed on the MC. See [About MC Privileges and Roles](#).

Manual Recovery

The process of recovery after an unclean shutdown of the database, where the administrator must accept recovery from the Last Good Epoch, which could lead to loss of some recently loaded data. See [Failure Recovery](#).

Mergeout

Mergeout is the Tuple Mover process that consolidates ROS containers and purges deleted records.

Meta-Functions

Used to query or change the internal state of Vertica and are not part of the SQL standard. See [Vertica Meta-functions](#) in the SQL Reference Manual.

Metadata

Data that describes the data in a database, such as data type, compression, constraints, and so forth, for each column in the tables. Metadata is stored in the Vertica catalog.

Moveout

The tuple mover operation of moving data from the WOS (Write Optimized Store), which is kept in memory, to the ROS (Read Optimized Store), which is kept on disk.

Multi-Homed

A computer host that has multiple IP addresses to connected networks.

NaN

An abbreviation for Not A Number, "NaN" (disregarding case) is optionally followed by a sequence of characters enclosed in parentheses. The character string specifies the value of NaN in an implementation-dependent manner. The Vertica internal representation of NAN, for example, is `0xfff8000000000000LL` on x86 machines.

Nibble

A four-bit aggregation (half a byte or half an octet) that can represent the decimal values 0 to 15 (16 distinct values). Since there are 16 possible values, a nibble corresponds to a single hexadecimal digit and is sometimes referred to as a "hex digit" or "hexit".

The following is a 4-bit nibble:

1 0 0 1

Note: In some systems, the term octet is used for an eight-bit unit instead of byte

Source: Wikipedia

Node

A host configured to run an instance of Vertica. It is a member of the database cluster. For a database to have the ability to recover from the failure of a node requires a database K-safety value of at least 1 (3+ nodes).

Node Definition

A metadata object that binds a host to a database. A node definition contains a symbolic node name that is used to specify segmentation in projections.

Nondeterministic

Nondeterministic functions could return different results each time they are called, even when the same input values are provided. For example, `GETDATE` and `CURRENT_TIMESTAMP` are nondeterministic functions.

Non-Repeatable Read

Occurs in a READ COMMITTED isolation level when two identical queries in the same transaction produce different results. This occurs when another transaction commits changes that alter the results of the query after the first query has completed and before the second query has begun.

NUL

Represents a character whose ASCII/Unicode code is zero, sometimes qualified "ASCII NUL".

NULL

Means *no value*, and is true of a field (column) or constant but not of a character.

Octal

The base-8 number system that uses eight unique symbols (0, 1, 2, 3, 4, 5, 6, and 7), where each digit represents three binary digits, starting from the right.

For example, the binary representation for decimal 74 is 0b1001010, which groups into 001 001 010 — so the octal representation is 112.

Whereas in decimal systems, each decimal place is a base of 10

$$74 = 7 \times 10^1 + 4 \times 10^0$$

in octal numerals, each place is a power with base 8:

$$112 = 1 \times 8^2 + 1 \times 8^1 + 2 \times 8^0$$

By performing the octal calculation, you can see 112 (octal) = 64 + 8 + 2 = 74 (in decimal).

Though octal is sometimes used in computing, instead of hexadecimal, hexadecimal is generally the more straightforward format and is used throughout the Vertica documentation.

Source: Wikipedia

Octet

A series of eight bits to form a single byte.

ODBC

An abbreviation for Open DataBase Connectivity, ODBC is a standard application programming interface (API) for access to database management systems.

OID

An object identifier (OID) is an identifier used to name an object. Structurally, an OID consists of a node in a hierarchically-assigned namespace, formally defined using the International Telecommunication Union-Telecommunication's (ITU-T) Abstract Syntax Notation standard (ASN.1). Vertica database objects are always assigned an OID. The OID can be used directly in some statements, such as [HAS_TABLE_PRIVILEGE](#).

Source: Wikipedia

OLAP

An abbreviation for Online Analytical Processing, OLAP answers multi-dimensional SQL analytical queries.

OLTP

An abbreviation for Online Transaction Processing, OLTP facilitates and manages real-time, transaction-oriented applications, such as data entry and retrieval transactions, for decision-support servers. OLTP technology is used in industries, such as financial services (e-trading and e-banking), manufacturing and mail order (e-commerce and order processing)

Out-Of-Date

A projection is out-of-date if it requires a refresh to participate in query execution.

Parallel Load

Parallel load is the process of loading data on any available node in the cluster (not necessarily the local node). Use this approach in combination with wildcards (such as *.dat) to load data files in parallel in a distributed manner. See COPY ON ANY NODE in COPY [COPY Parameters](#)

Path (quality plan)

The execution strategy of the Vertica cost-based query optimizer, denoting a sub operation in the query plan.

Partition Key

A table column that specifies how the table is partitioned.

Partitioning

Specifies how data is organized within individual nodes. Partitioning attempts to introduce hot spots within the node, providing a convenient way to drop data and reclaim the disk space.

Note: Partitioning and segmentation are terms often used interchangeably for other databases, but they have completely separate goals in Vertica regarding data localization. See [Comparing Partitioning and Segmentation](#) in the Administrator's Guide for details.

Phantom Read

Occurs in a READ COMMITTED isolation level when two identical queries in the same transaction produce different collections of rows. This occurs because a table lock is not acquired on SELECT during the initial query.

Physical Schema

Consists of a set of projections used to store data on disk. The projections in the physical schema are based on the objects in the logical schema.

Physical Schema Design

A usable K=1 design based on an analysis of the sample data files and queries (if available). The physical schema design contains segmented (fact table) superprojections and replicated (dimension table) superprojections.

Predicate

Specify conditions that can be evaluated to SQL [three-valued logic](#) (3VL) Boolean truth values. Predicates are also used to limit the effects of statements and queries.

Primary Column

A column that is loaded from raw data, and not derived from an expression.

Primary Key

A single column or combination of columns (called a compound key) that uniquely identifies each row in a table. A PRIMARY KEY constraint contains unique, non-null values.

Projection

Optimized collections of table columns that provide physical storage for data. A projection can contain some or all of the columns of one or more tables.

For conceptual information about Vertica projections, see [Physical Schema](#) in Vertica Concepts. For information about using and managing projections, see [Working with Projections](#) in the Administrator's Guide.

Projection set

A group of buddy projections that are safe for a given level of K-safety. When $K=1$, there are two buddies in a set; when $K=2$, there are three buddies. The Database Designer assigns all projections in a projection set the same base name so they can be identified as a group.

A projection must be part of a projection set before it is refreshed. Once a projection set is created (by creating buddies), the set is refreshed in a single transaction.

Purge

Permanently removes deleted data from physical storage so disk space can be reused. You can purge historical data up to and including the Ancient History Mark epoch.

Query Cluster Level

Determines the number of sets used to group similar queries. The query cluster level can be any integer from one (1) to the number of queries to be included in the physical schema design.

Queries are generally grouped based on the columns they access and the way in which they are used. The following work loads typically use different types of queries and are placed in different query clusters: drill downs, large aggregations, and large joins. For example, if a reporting tool and dashboard both access the same database, the reporting tool is likely to use a drill down to access a subset of data and the dashboard is likely to use a large aggregation to look across a large range of data. In this case, there would be at least two (2) query clusters.

Query Optimizer

The component that evaluates different strategies for running a query and picks the best one.

RDBMS

An abbreviation for relational database management system.

RLE (Run Length Encoding)

The abbreviation for Run Length Encoding, RLE replaces sequences of the same data values within a column by a single value and a count number.

Recovery

Vertica can restore the database to a fully functional state after one or more nodes in the system experiences a software- or hardware-related failure. Vertica recovers nodes by querying replicas of the data stored on other nodes. For example, a hardware failure can cause a node to lose database objects or to miss changes made to the database (INSERTs, UPDATEs, and so on) while offline. When the node comes back online, queries other nodes in the cluster to recover lost objects and catch up with database changes.

Referential Integrity

Consists of a set of constraints (logical schema objects) that define primary key and foreign key columns.

- Each small table must have a PRIMARY KEY constraint.
- The large table must contain columns that can be used to join the large table to smaller tables.
- Outer join queries produce the same results as the corresponding inner join query if there is a FOREIGN KEY constraint on the outer table. Note that the inner table of the outer join query must always have a PRIMARY KEY constraint on its join columns.

Refresh

Ensures that all projections on a node are up-to-date (can participate in query execution). This process could take a long time, depending on how much data is in the table(s).

Replication

The Vertica process of storing identical copies of data across all nodes in a cluster.

Replay Attack

Occurs when the same hash (password equivalent value) can be sent by a different source and successfully authenticated. Vertica uses timestamps to prevent hashing because a timestamp is unique for each authentication request. For example, all passwords are combined with the user name and current timestamp for each authentication request.

Resegmentation

A process that Vertica performs automatically during query execution that distributes the rows of an existing projection or intermediate relation evenly to each node in the cluster. At the end of resegmentation, every row from the input relation is on exactly one node. Vertica resements data when the input does not have the segmentation required to compute the requested result efficiently and correctly.

Resource Pool

A resource pool comprises a pre-allocated subset of the system resources, with an associated queue. A resource pool is created using the `CREATE RESOURCE POOL` command as described in the SQL Reference Manual.

Resource Manager

In a single-user environment, the system can devote all resources to a single query and get the most efficient execution for that one query. However, in a environment where several concurrent queries are expected to run at once, there is tension between providing each query the maximum amount of resources (thereby getting fastest run time for that query) and serving multiple queries simultaneously with a reasonable run time. The Resource Manager (RM) provides options and controls for resolving this tension, while ensuring that every query eventually gets serviced and that true system limits are respected at all times.

Role

A role groups together a set of privileges that can be assigned to a user or another role. You can use roles to quickly grant or revoke privileges on multiple tables, schemas, functions or other database entities to one or more users with a single command.

Vertica's implementation of roles conforms to the SQL Standard T331 for basic roles.

Rollback

Transaction rollbacks restore a database to an earlier state by discarding changes made by that transaction. Statement-level rollbacks discard only the changes initiated by the reverted statements. Transaction-level rollbacks discard all changes made by the transaction.

With a ROLLBACK statement, you can explicitly roll back to a named savepoint within the transaction, or discard the entire transaction. Vertica can also initiate automatic rollbacks in two cases:

- An individual statement returns an ERROR message. In this case, Vertica rolls back the statement.
- DDL errors, systemic failures, dead locks, and resource constraints return a ROLLBACK message. In this case, Vertica rolls back the entire transaction.

Explicit and automatic rollbacks always release any locks that the transaction holds.

ROS (Read Optimized Store)

Read Optimized Store (ROS) is a highly optimized, read-oriented, disk storage structure, organized by projection. ROS makes heavy use of compression and indexing.

ROS container

A ROS (Read Optimized Store) container is a set of rows stored in a particular group of files. ROS containers are created by operations like Moveout or COPY DIRECT. You can query the STORAGE_CONTAINERS system table to see ROS containers. The ROS container layout can differ across nodes due to data variance. Segmentation can deliver more rows to one node than another. Two data loads could fit in the WOS on one node and spill on another.

RPM

A powerful package manager, which can be used to build, install, query, verify, update, and erase individual software packages. A package consists of an archive of files and metadata used to install and erase the archive files. The metadata includes helper scripts, file attributes, and descriptive information about the package. Packages come in two varieties: binary packages, used to encapsulate software to be installed, and source packages, containing the source code and recipe necessary to produce binary packages.

Safe

A projection is safe if it has enough buddy projections for the current K-safety level.

SAN

An abbreviation for Storage Area Network, SAN is a dedicated hardware/software platform consisting of networked servers that provide access to storage-related resources such as such as disk array controllers. It provides high data transfer speeds similar to those used for internal disk drives (ATA, SCSI, etc.) and is highly scalable.

Salt

Random bytes included in the password hash to prevent replay attacks. See [Implementing Client Authentication](#) in the Administrator's Guide.

Savepoint

A *savepoint* is a special marker inside a transaction that allows commands that execute after the savepoint to be rolled back. The transaction is restored to the state that preceded the savepoint.

Vertica supports two types of savepoints:

- An *implicit savepoint* is automatically established after each successful command within a transaction. This savepoint is used to roll back the next statement if it returns an error. A transaction maintains one implicit savepoint, which it rolls forward with each successful command. Implicit savepoints are available to Vertica only and cannot be referenced directly.
- *Named savepoints* are labeled markers within a transaction that you set through [SAVEPOINT](#) statements. A named savepoint can later be referenced in the same transaction through [RELEASE SAVEPOINT](#), which destroys it, and [ROLLBACK TO SAVEPOINT](#), which rolls back all operations that followed the savepoint. Named savepoints can be especially useful in nested transactions: a nested transaction that begins with a savepoint can be rolled back entirely, if necessary.

Schema

Has several related meanings in Vertica:

- In SQL statements, a schema refers to named namespace for a logical schema.
- Logical schema refers to a set of tables and constraints.
- Physical schema refers to a set of projections.

Secure Shell

A set of standards and an associated network protocol that establishes a secure TCP/IP data transmission channel between a local and a remote computer. Secure Shell (SSH) utilizes strong encryption and authentication to ensure confidentiality, integrity, and authenticity of the transferred data.

SSH is typically used to log in to a remote machine and run commands. Use SSH only between two devices that are both under your own administration, when both devices are trustworthy.

Seed

The starting value used by a random number generation routine to create random numbers.

Segmentation

Defines how physical data storage (projections) is stored in a database cluster using the [CREATE PROJECTION](#) statement. The goal is to distribute data evenly across multiple nodes in the database so that all nodes can participate in query execution.

Note: Partitioning and segmentation are terms often used interchangeably for other databases, but they have completely separate goals in Vertica regarding data localization. See [Comparing Partitioning and Segmentation](#) in the Administrator's Guide for details.

Select List

The list of expressions that appears after the `SELECT` keyword and before the `FROM` clause is called the select list. The select list is where you specify the columns in the set of records you want Vertica to return from one or more tables or views.

Session

An occurrence of a user interacting with a database through the use of SQL statements. You can start a session using vsql or a JDBC application. In Vertica, the scope of a session is the same as that of a connection. Connecting to the database starts a session, and exiting ends it.

Session-scoped data is preserved beyond the lifetime of a single transaction. Terminating a session truncates a table and deletes all rows.

Site

A deprecated term in Vertica that is used to mean node or host in some contexts, such as the CREATE PROJECTION statement.

Snapshot Isolation

Vertica can run any SQL query in snapshot isolation mode in order to obtain the fastest possible execution. To be precise, snapshot isolation mode is actually a form of a historical query. The syntax is:

```
AT EPOCH LATEST SELECT...
```

The command queries all data in the database up to but not including the current epoch without holding a lock or blocking write operations, which could cause the query to miss rows loaded by other users up to (but no more than) a specific number of minutes before execution.

Snowflake Schema

The same as a star schema except that a dimension table can be normalized (hierarchically decomposed) into additional dimension tables. Every dimension table participates in a 1::n join with the fact table or another dimension table.

Spread

An open source toolkit used in Vertica to provide a high performance messaging service that is resilient to network faults. Spread daemons start automatically when a database starts up for the first time, and the spread process runs on [control nodes](#) in the cluster.

Sort-Merge Join

If both inputs are pre-sorted, merge joins do not have to do any pre-processing. Vertica uses the term sort-merge join to refer to the case when one of the inputs must be sorted prior to the merge join. Vertica sorts the inner input side but only if the outer input side is already sorted on the join keys.

Source (table)

In [MERGE](#) operations, source is the table reference that contains the new and/or changed data you want to use to update the target table.

SQL

An abbreviation for Structured Query Language, SQL is a widely-used, industry standard data definition and data manipulation language for relational databases.

SSH

An abbreviation for Secure Shell, SSH is a set of standards and an associated network protocol that establishes a secure TCP/IP data transmission channel between a local and a remote computer. Secure Shell (SSH) uses strong encryption and authentication to ensure confidentiality, integrity, and authenticity of the transferred data.

SSH is typically used to log in to a remote machine and run commands. Use SSH only between two devices that are both under your own administration, when both devices are trustworthy.

Stable functions

When run with a given set of arguments, stable functions produce the same result within a single query or scan operation. However, a stable function can produce different results when issued under different environments or at different times, such as change of locale and time zone—for example, `SYSDATE()` and `'today'`.

Star Schema

Sometimes called a star join schema, a star schema is the simplest data warehouse schema. In a star schema design there is a central fact table with a large number of records, optionally surrounded by a collection of dimension tables, each with a lesser number of records. Every dimension table participates in a 1::n join with the fact table.

Standalone Resource Pool

A pool that is configured such that it cannot borrow memory from the GENERAL pool. A standalone pool has MEMORYSIZE = MAXMEMORYSIZE.

Storage Location

A directory path used by Vertica to store catalog, actual data files, and temporary data files.

You can also create storage locations for users, and then grant one or more users access to the storage. You can also create a storage location with a location label, for use in storage policies.

Storage Policy

A database object you create to associate a labeled location as the default storage location for the object. Database object can be a database, schema, table, or min- and max- ranges of a partition.

Strict

Indicates that a function always returns null when any of its input arguments is null.

Superprojection

A projection that includes all columns in an anchor table. Vertica uses superprojections to ensure support for all queries and other DML operations.

Under certain conditions, Vertica [automatically creates a table's superprojection](#) immediately on table creation. Vertica also creates a superprojection when you first load data into that table, if none already exists. `CREATE PROJECTION` can create a superprojection if it specifies to include all table columns. A table can have multiple superprojections.

Superuser

The automatically-created database user who has the same name as the Linux database administrator account and who can bypass all GRANT/REVOKE authorization, or any user that has been granted the PSEUDOSUPERUSER role. Do not confuse the concept of a database superuser with Linux superuser (root) privilege. A database superuser cannot have Linux superuser privilege.

TM

An abbreviation for the [Tuple Mover](#).

Table

In the relational model, a table (relation) is a set of data elements (values) organized into horizontal rows (tuples) and vertical columns. A table has a specified number of columns but can have any number of rows.

In Vertica, a table is a metadata-only entity referred to in queries. The physical representation of data is a projection.

Target (table)

In [MERGE](#) operations, the target table is the table that contains the existing records you want to update, using records from the source table.

Temp Location

A storage location used as temp space by Vertica.

Temp Space

Disk space temporarily occupied by temporary files created by certain query execution operations, such as hash joins and sorts, in the case when they have to spill to disk. Such operations might also be encountered during queries, recovery, refreshing projections, and so on. If a temp location is provided to the database, Vertica uses it as temp space.

Top-K Projection

A projection that configures the projection data for a Top-K query. A Top-K query is one that retrieves the top k rows from a group of tuples.

A Top-K projection is a type of live aggregate projection.

Transaction

One or more operations that are executed as a unit of work. At the user level, transactions occur in the current session by a user or script running one or more SQL statements. When you commit a transaction, any changes you make to data in tables using `INSERT`, `DELETE`, `UPDATE`, `MERGE`, and `COPY` during the transaction become permanent. If you roll back the transaction, all changes made to table data are undone.

Vertica supports Atomicity, Consistency, Isolation, and Durability (ACID) for SQL transactions.

Tuple

A collection of data values corresponding conceptually to a row of a table, projection, or result set. A tuple in Vertica does not have the same physical representation as in a traditional RDBMS; it is a virtual entity that could have entirely different storage representations.

Tuple Mover

The Tuple Mover manages WOS and ROS data storage. It performs two operations:

- [Moveout](#) moves data from WOS to ROS. During moveout operations, the Tuple Mover also enforces storage policies for the storage location.
- [Mergeout](#) combines small ROS containers into larger ones and purges deleted data.

The Tuple Mover automatically performs these tasks in the background, at intervals that are set by its [configuration parameters](#).

UDx

User-Defined Extensions (UDxs) are functions contained in external shared libraries that are developed in C++, Python, Java, or R using the Vertica SDK. The external libraries are defined in the Vertica catalog using the [CREATE LIBRARY](#) statement. They are best suited for analytic operations that are difficult to perform in SQL, or need to be performed frequently enough that their speed is a major concern.

UDP

An abbreviation for User Datagram Protocol, UDP is an Open Systems Interconnection (OSI) transport layer protocol for client/server network applications based on Internet Protocol (IP). UDP is the primary alternative to the Transmission Control Protocol (TCP) and is used to send messages (datagrams), to other hosts on an IP network. UDP messages can be sent to one host, a group of hosts (multicast), or all hosts (broadcast) on an IP subnet.

User-Defined SQL Function

User-Defined SQL Functions let you define and store commonly-used SQL expressions as a function. User-Defined SQL Functions are useful for executing complex queries and combining Vertica built-in functions. You simply call the function name you assigned in your query.

A User-Defined SQL Function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

UTC

An abbreviation for Coordinated Universal Time (in English), UTC is the high-precision atomic time standard that replaced Greenwich Mean Time on 1 January 1972 as the basis for the main reference time scale or civil time in various regions. UTC is also referred to by the military and civil aviation as Zulu time (Z).

UTF-8

Vertica supports Unicode Transformation Format-8, or UTF8, where 8 equals 8-bit. UTF-8 is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard. Initial encoding of byte codes and character assignments for UTF-8 coincides with ASCII. Thus, UTF8 requires little or no change for software that handles ASCII but preserves other values.

Vertica database servers expect to receive all data in UTF-8, and Vertica outputs all data in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. JDBC and ADO.NET APIs operate on data in UTF-16. Client drivers automatically convert data to and from UTF-8 when sending to and receiving data from Vertica using API calls. The drivers do not transform data loaded by executing a [COPY](#) or [COPY LOCAL](#) statement.

Unary Operator

An operation with a single input.

Up to Date

See Up-to-date.

Up-To-Date

A projection is up-to-date (or up to date) if it is eligible to participate in query execution. Projections on empty tables are up-to-date upon creation. If the table has data loaded already, newly created projections are marked not up-to-date until refreshed. If a projection is refreshed while a node is down, that projection can be marked up-to-date even though it is missing data on one of the nodes. This is because the node will build the data during the recovery process before participating in queries.

User Agent

A client application program used to access resources on networks such as the World Wide Web. User agents include web browsers, search engine crawlers, PDAs, cell phones, and so forth.

View

A named logical relation specified by an associated query that can be accessed similarly to a table in the FROM clause of a SQL statement. The results of the query are not stored but obtained on the fly when the SQL referencing the view is executed.

Volatile functions

Regardless of their arguments or environment, volatile functions can return a different result with each invocation—for example, [RANDOM\(\)](#).

Volatility

Indicates whether a function returns the same output given the same input. There are 3 types:

- Immutable (same output given same input)
- Stable (same output within the same row)
- Volatile (different output for each invocation, such as `RANDOM()`)

vsq!

vsq! is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.

WLA

An abbreviation for Workload Analyzer.

Webhook

A subscription that the Management Console application/web server sends to each agent in the Vertica cluster. MC creates a webhook when it connects to the agent, and the agent stores this information in local memory and on disk.

When MC executes a long-running job on an agent, such as a create-database operation, the agent uses the webhook subscription to send messages back to MC. If an agent on a cluster node stops while the MC is monitoring the database cluster during a long-running job, MC discovers that the agent is down and switches to another agent in that cluster.

Window (analytic)

An analytic function's `OVER` clause specifies how to partition, sort, and frame function input with respect to the current row. The input data is the result set that the query returns after it evaluates `FROM`, `WHERE`, `GROUP BY`, and `HAVING` clauses.

Workload Analyzer

An advisor tool that analyzes system information held in [SQL system tables](#) (monitoring APIs) and returns a set of tuning recommendations. See [Analyzing Workloads](#) in the Administrator's Guide for details.

WOS (Write Optimized Store)

Write Optimized Store (WOS) is a memory-resident data structure for short-term data storage. It is typically accessed by data manipulation language (DML) statements that load or remove data: [INSERT](#), [COPY](#), [UPDATE](#), and [DELETE](#). To support very fast data load speeds, WOS stores records without data compression or indexing. A projection in WOS is sorted only when it is queried. It remains sorted as long as no further data is inserted into it. WOS organizes data by epoch and holds both committed and uncommitted transaction data.

Third-Party Software Acknowledgements

This document contains the licenses and notices for open source software used in Vertica Analytics Platform 8.1.x.

You can download open source code from <https://my.vertica.com/licenses/>.

Micro Focus makes no representations or warranties regarding any third party software. All third-party software is provided or recommended by Vertica on an AS IS basis.

This product includes cryptographic software written by Eric Young (ey@cryptsoft.com).

Requesting Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by EntIT Software LLC. To obtain such source code, send a check or money order in the amount of US \$10.00 to:

EntIT Software LLC
Attn: General Counsel
1140 Enterprise Way
Sunnyvale, CA 94089
USA

Please specify the product and version for which you are requesting source code.

Angularjs

Version 1.3, Copyright (c) 2010-2015 Google, Inc. <https://angularjs.org> and <http://webjars.org>

A Webjar for AngularJS.

MIT License (MIT) Copyright (c) 2016 Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Angular-bootstrap

Bootstrap widgets for Angular homepage: <http://angular-ui.github.io/bootstrap>.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Angular-multi-step-form

Angular module for creating wizards and multi-step forms, and licensed under ISC License (ISC):

<https://github.com/troch/angular-multi-step-form>

Copyright © 2004-2013 by Internet Systems Consortium, Inc. ("ISC")

Copyright © 1995-2003 by Internet Software Consortium

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Angular-ui-grid

Angular UI grid for use in Vertica Management Console (MC),
<https://www.nuget.org/packages/angular-ui-grid>.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Apache-Arrow

Version 0.2.0

Apache License 2.0

Apache-arrow 0.2.0 is an early release and the APIs are still evolving.

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or

other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Avalon Framework

Copyright © 1999- 2004, Apache Software Foundation.

All rights reserved.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Avro

Version 1.7.0

Copyright © 2013 The Apache Software Foundation.

Apache Hadoop, Hadoop, HDFS, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Zookeeper are trademarks of the Apache Software Foundation.

<http://avro.apache.org/>

Description

Data serialization system: Apache Avro is a data serialization system. Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages.

Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

Declared Fedora Licences: ASL 2.0

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any

separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Commons Collections

Copyright © 2001-2008 The Apache Software Foundation

The Java Collections Framework was a major addition in JDK 1.2. It added many powerful data structures that accelerate development of most significant Java applications. Since that time it has become the recognized standard for collection handling in Java. Commons-Collections seek to build upon the JDK classes by providing new interfaces, implementations and utilities. For more information, see <https://commons.apache.org/proper/commons-collections/>.

License Acknowledgments

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Commons Configuration

Copyright © 2001-2012 The Apache Software Foundation. All Rights Reserved

Version 1.6

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Commons DBCP

Copyright © 2001-2014 The Apache Software Foundation. All Rights Reserved.

Version 1.4

Version 2.0.1

DBCP 2 binaries should be used by applications running under Java 7.

DBCP 1.4 binaries should be used by applications running under Java 6.

License Acknowledgments

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Commons FileUpload

Copyright © 2002-2010 The Apache Software Foundation. All Rights Reserved.

Version 1.2

The Commons FileUpload package makes it easy to add robust, high-performance, file upload capability to your servlets and web applications.

License Acknowledgments

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Commons Logging

Copyright © 2001-2008 The Apache Software Foundation

The Apache Commons Logging component is a thin adapter allowing configurable bridging to other, well-known logging systems.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Derby

Copyright © 2004-2012 Apache Software Foundation

Version 10.6.2.1

Apache Derby, an Apache DB subproject, is an open source relational database, implemented entirely in Java. Vertica uses Apache Derby in Management Console (MC).

License Acknowledgments

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Hadoop

Copyright © 2012 The Apache Software Foundation.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Hadoop Libhdfs++

Apache License 2.0, Apache component version 2.7.2.

Vertica provides the libhdfs++ library to support C++ calls to the Apache Hadoop Distributed File System (HDFS). This version has new functionality and substantial modifications.

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or

other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Hadoop Libhdfs++ Kerberos

Apache License 2.0, Apache component version 2.7.2

Vertica provides Kerberos support for the libhdfs++ library to support C++ calls to the Apache Hadoop Distributed File System (HDFS). This version has new functionality and substantial modifications.

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Hive

Copyright © 2013 The Apache Software Foundation.

Apache Hadoop, Hadoop, HDFS, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Zookeeper are trademarks of the Apache Software Foundation.

<http://hive.apache.org/>

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache ORC

Apache License 2.0, Copyright © 2004

All rights reserved.

Description

The smallest, fastest columnar storage for Hadoop workloads.

Apache License Version 2.0

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache ORC Subcomponents

The Apache ORC project contains subcomponents with separate copyright notices and license terms. Your use of the source code for these subcomponents is subject to the terms and conditions of the following licenses.

Parts of the site formatting are licensed under the MIT License (MIT):

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Apache Parquet

Apache License 2.0, Parquet Component 2.3

Apache Parquet is a general-purpose columnar storage format, built for Hadoop. You can use Parquet with any data processing framework, data model, or programming language. The Apache Parquet library lets Vertica users read Parquet files efficiently. This work includes building an optimized reader in C++ for parquet files.

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

ASMJIT

Copyright © 2008-2010, Petr Kobalíček <kobalicek.petr@gmail.com>

All rights reserved.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

asm.asm

Copyright © 2000-2011 INRIA, France Telecom

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

asm.asm-commons

Copyright © 2000-2011 INRIA, France Telecom

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

asm.asm-tree

Copyright © 2000-2011 INRIA, France Telecom

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Atinject 1

Copyright (C) 2009. The JSR-330 Expert Group.

Version 1

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

AWS-AWS-sdk-cpp

Version 1.0.34

Apache License 2.0

The Amazon AWS SDK for C++, <https://github.com/aws/aws-sdk-cpp>.

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or

other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

BLAS

Version 3.6.0, updated November 2015

Basic Linear Algebra Subprograms (BLAS) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

License and Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. ASC-9313958 and DOE Grant No. DE-FG03-94ER25219. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF) or the Department of Energy (DOE).

The reference BLAS is a freely-available software package. It is available from netlib via anonymous ftp and the World Wide Web. Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors.

Like all software, it is copyrighted. It is not trademarked, but we do ask the following:

- If you modify the source for these routines we ask that you change the name of the routine and comment the changes made to the original.
- We will gladly answer any questions regarding the software. If a modification is done, however, it is the responsibility of the person who modified the routine to provide support.

Boost

Boost provides free peer-reviewed portable C++ source libraries. For more information, see www.boost.org.

Version 1.59 - August 13th, 2015

Version 1.38 - February 8th, 2009

Copyright Beman Dawes, David Abrahams, 1998-2005.

Copyright Rene Rivera 2004-2005.

Boost Software License

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Bootstrap

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile, first projects on the web.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Bottle

Copyright © 2011, Pierre Quentel (pierre.quentel@gmail.com)

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Brotli

Brotli is a Google data encoding and compression format, used primarily to improve performance in modern browsers.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

bzip2

Copyright © 1996-2005 Julian R Seward

This file is a part of bzip2 and/or libbzip2, a program and library for lossless, block-sorting data compression.

Copyright © 1996-2005 Julian R Seward. All rights reserved.

1. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
2. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
3. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
4. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
5. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, Cambridge, UK.

jseward@bzip.org <<mailto:jseward@bzip.org>>

bzip2/libbzip2 version 1.0 of 21 March 2000

This program is based on (at least) the work of:

Mike Burrows

David Wheeler

Peter Fenwick

Alistair Moffat

Radioed Neal

Ian H. Witten

Robert Sedgewick

Jon L. Bentley

cmake

Cross Platform Makefile Generator

Copyright 2000-2009 Kitware, Inc., Insight Software Consortium

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



Version 7.29.0

Copyright (c) 1996 - 2016, Daniel Stenberg, <daniel@haxx.se>.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

curl_fopen.c

Copyright (c) 2003 Simtec Electronics

Re-implemented by Vincent Sanders<vince@kyllikki.org> with extensive reference to original curl example code

- Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

cyrus-sasl-library

Version: 2.1.26

For Vertica SQL on Hadoop, this is the Cyrus SASL API implementation. Use this library on the client or server side to provide authentication. This software package contains encryption software. Be sure to abide by appropriate export rules if you download it.

For more information, see <http://asg.web.cmu.edu/sasl/sasl-library.html>.

Description

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



Copyright 2010 - 2016 Michael Bostock

Version 3

License Acknowledgments

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Daemonize

Copyright © 2003-2011

Brian M. Clapper. All rights reserved

This software runs a command as a Unix daemon.

License Acknowledgments

With the exception of the install-sh script and the getopt.c source, this software is released under BSD license, which follows:

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Derby

Copyright © Apache Software Foundation

Version 10.6.1.0

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Derbyclient

Copyright © Apache Software Foundation

Version 10.2.2.0

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Derbynet

Copyright © Apache Software Foundation

Version 10.6.2.1

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Flex Extractor Functions

Libraries to support extracting flex Vmaps include:

- MAPJSONEXTRACTOR
- MAPDELIMITEDEXTRACTOR
- MAPREGEXEXTRACTOR

License

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any

character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Flex Parsers

Parsers for unstructured data for flex and columnar tables are as follows:

- Avro
- CEF
- CSV
- Delimited
- JSON
- RegEx

License

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,

MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Flex Vmap Functions

Libraries to perform functions on Vmaps:

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPTOSTRING
- MAPVALUES
- MAPVERSION

License

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any

separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Flex Vmaps

Table column format able to hold unstructured data. Vmap data does not require a schema definition.

License

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of

this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or

malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Font Awesome

Copyright © 2014 Font Awesome by Dave Gandy - <http://fontawesome.io>

Version 4.3.0

MIT License (MIT) Copyright (c) 2016 Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

FreeMarker

Apache FreeMarker is a template engine: a Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data. For MC, FreeMarker is used to develop the threshold notification email template.

License Information

Copyright 2014 Attila Szegedi, Daniel Dekany, Jonathan Revusky

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

=====
=

FreeMarker Subcomponents with Different Copyright Owners

FreeMarker, both in its source code and binary form (freemarker.jar) includes a number of files that are licensed by the Apache Software Foundation under the Apache License, Version 2.0. This is the same license as the license of FreeMaker, but the copyright owner is the Apache Software Foundation. These files are:

- freemarker/ext/jsp/web-app_2_2.dtd
- freemarker/ext/jsp/web-app_2_3.dtd
- freemarker/ext/jsp/web-jsptaglibrary_1_1.dtd
- freemarker/ext/jsp/web-jsptaglibrary_1_2.dtd

Historical Notes

FreeMarker 1.x was released under the LGPL license. Later, by community consensus, we have switched over to a BSD-style license. As of FreeMarker 2.2pre1, the original author, Benjamin Geer, has relinquished the copyright in behalf of Visigoth Software Society. With FreeMarker 2.3.21 the license has changed to Apache License, Version 2.0, and the owner has changed from Visigoth Software Society to three of the FreeMarker 2.x developers, Attila Szegedi, Daniel Dekany, and Jonathan Revusky.

Ganglia Open Source License

Copyright © 2001 by Matt Massie and The Regents of the University of California.

All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without written agreement is hereby granted, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GEOS

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the GEOS package can be found at <http://my.vertica.com/licenses/geos-3.3.8.tar.bz2>.

gperftools-libs

Version: 2.1

(Originally, Google Performance Tools), gperftools is a collection of a high-performance multi-threaded malloc() implementation, described as a thread-friendly heap-checker, heap-profile, and cpu-profiler. Includes tcmalloc, libtcmalloc, and libprofiler.

Description

Copyright (c) 2005, Google Inc.

All rights reserved.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Gradle-Node-Plugin

This is the Gradle plugin for running NodeJS scripts.

Copyright © Apache Software Foundation

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Guava Libraries

Version 18.0

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

HCatalog

Version 0.5.0

Copyright © 2012-2013 The Apache Software Foundation

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Hibernate Validator

Copyright © 2009 - 2013 Red Hat, Inc. & Gunnar Morling

Version 5.1.3

<http://www.apache.org/licenses/Version 2.0, January 2004> TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions. "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document. "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License. "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity. "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License. "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files. "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types. "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below). "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof. "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution." "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a

perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions: You must give any other recipients of the Work or Derivative Works a copy of this License; and You must cause any modified files to carry prominent notices stating that You changed the files; and You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as

required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.END OF TERMS AND CONDITIONS

html-minifier

Version 3.3.1

<https://www.npmjs.com/package/html-minifier>

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

HttpClient

Copyright Apache Software Foundation

Version 4.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Httpcore

Copyright Apache Software Foundation

Version 4.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

httplib2

Copyright © 2006, Joe Gregorio (joe@bitworking.org)

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Httpmime

Copyright © Apache Software Foundation

Version 4.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

ICU (International Components for Unicode) License - ICU 1.8.1 and Later

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1995-2009 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

JSON.simple

Copyright © 2009, Yidong Fang, Chris Kobleberg

A simple Java toolkit for JSON.

License Acknowledgments

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of

this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or

malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

javax.el.el-api

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. “Contributor” means each individual or entity that creates or contributes to the creation of Modifications.

1.2. “Contributor Version” means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. “Covered Software” means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. “Executable” means the Covered Software in any form other than Source Code.

1.5. “Initial Developer” means the individual or entity that first makes Original Software available under this License.

1.6. “Larger Work” means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. “License” means this document.

1.8. “Licensable” means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. “Modifications” means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. “Original Software” means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. “Patent Claims” means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. “Source Code” means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. “You” (or “Your”) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, “You” includes any entity which controls, is controlled by, or is under common control with You. For purposes of this

definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients' rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient's rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3 Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN “AS IS” BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as “Participant”) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY’S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS

SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” (as that term is defined at 48 C.F.R. § 252.227-7014(a)(1)) and “commercial computer software documentation” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction’s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys’ fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to

the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections

when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However,

parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those

countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You are entitled to receive the source code for such software. For no less than three years from the date you obtained this software package, you may download a copy of the source code for the software in this package licensed under the GPL at no charge by visiting:

<http://my.vertica.com/licenses/el-api-1.1.jar>

javax.inject

Copyright 2001-2012 Bob Lee

Version 1

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

javax.transaction.jta

Copyright © 2002 Sun Microsystems, Inc. All rights reserved.

Java(TM) Transaction API (JTA) Specification ('Specification')

Version: 1.0.1B

Status: Maintenance Release

Release: November 5, 2002

Copyright 2002 Sun Microsystems, Inc.

4150 Network Circle, Santa Clara, California

95054, U.S.A

All rights reserved.

NOTICE; LIMITED LICENSE GRANTS

Sun Microsystems, Inc. ('Sun') hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under the Sun's applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation, which shall be understood to include developing applications intended to run on an implementation of the Specification provided that such applications do not themselves implement any portion(s) of the Specification.

Sun also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or patent rights it may have in the Specification to create and/or distribute an Independent Implementation of the Specification that: (i) fully implements the Spec(s) including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and (iii) passes the TCK (including satisfying the requirements of the applicable TCK Users Guide) for such Specification. The foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose.

You need not include limitations (i)-(iii) from the previous paragraph or any other particular 'pass through' requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to implementations of the Specification (and products derived from them) that satisfy limitations (i)-(iii) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Sun's applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation's compliance with the Spec in question.

For the purposes of this Agreement: 'Independent Implementation' shall mean an implementation of the Specification that neither derives from any of Sun's source code or binary code materials nor, except with an appropriate and separate license from Sun, includes any of Sun's source code or binary code materials; and 'Licensor Name Space' shall mean the public class or interface declarations whose names begin with 'java', 'javax', 'com.sun' or their equivalents in any subsequent naming convention adopted by Sun through the Java Community Process, or any recognized successors or replacements thereof.

This Agreement will terminate immediately without notice from Sun if you fail to comply with any material provision of or act outside the scope of the licenses granted above.

TRADEMARKS

No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED 'AS IS'. SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or clean room implementation; and/or (iii) any claims that later

versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your use of the Specification ('Feedback'). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

Jdom

Version 1.1.3

Copyright © 2011 Jason Hunter, Brett McLaughlin. All Rights Reserved.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

jemalloc

Version 4.2.1

This software is the jemalloc at github.com. See <http://www.canonware.com/jemalloc/>

Open Source Component jemalloc

Unless otherwise specified, files in the jemalloc source distribution are subject to the following license:

Copyright (C) 2002-2016 Jason Evans <jasone@canonware.com>.
All rights reserved.

Copyright (C) 2007-2012 Mozilla Foundation. All rights reserved.

Copyright (C) 2009-2016 Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

BSD 2-clause "Simplified" License

All rights reserved.

See <http://spdx.org/licenses/BSD-2-Clause>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Jetty

Copyright ©, Eclipse Foundation

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the

Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under

this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

jquery

Copyright © 2011 jQuery Team

All rights reserved.

Version 2.1.0

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Jquery-Selectmenu

Copyright © 2011, Paul Bakaus, <http://jqueryui.com/>

This software consists of voluntary contributions made by many individuals (AUTHORS.txt, <http://jqueryui.com/about>) For exact contribution history, see the revision history and logs, available at <http://jquery-ui.googlecode.com/svn/>

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Jquery-Sparklines

Copyright © 2011, jQuery Team

BSD 2-clause "Simplified" License

All rights reserved.

See <http://spdx.org/licenses/BSD-2-Clause>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

jQuery Steps

Use the jQuery Wizard widget for MC Data Ingest tool.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

junitjs

Copyright © 2011 JS Foundation

<https://js.foundation/>

Version 1.20.0

JavaScript unit testing framework, <https://qunitjs.com/>.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

json-cpp

Version 0.6.0-rc2

Copyright (c) 2007-2010 Baptiste Lepilleur

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

json-org-java

Version 20140107

Copyright (c) 2002 JSON.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

junitjs

Copyright © 2011 JS Foundation

<https://js.foundation/>

Version 1.20.0

JavaScript unit testing framework, <https://qunitjs.com/>.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



Copyright (C) 2000-2007 the JWNL development team
(<http://www.sourceforge.net/projects/jwordnet>)

All rights reserved.

Version 1.3.3

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Karma

Spectacular Test Runner for JavaScript

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-Chrome-Launcher

A Karma plugin. Launcher for Chrome and Chrome Canary.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-Coverage

A Karma plugin to generate code coverage.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-Firefox-Launcher

A Karma plugin launcher for Firefox.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-Jasmine

A Karma plugin - adapter for Jasmine testing framework.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-Junit-Reporter

A Karma plugin to report results in junit xml format.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-ng-html2js-preprocessor

A Karma plugin. Adapter for QUnit testing framework.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-Phantomjs-Launcher

A Karma plugin. Launcher for PhantomJS.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Karma-qunit

A Karma plugin. Adapter for QUnit testing framework.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

language-detection

Copyright (c) 2010-2011 Cybozu Labs, Inc. All rights reserved.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

LAPACK

Version: 3.4.0

Updated November 11, 2011

LAPACK is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision. Vertica statically links this to other code (compiled code co-mingled in the same binary files).

Description

BSD 3-clause "New" or "Revised" License

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Libcsv

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the libcsv dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at <http://my.vertica.com/licenses/libcsv-3.0.1.tar.gz>

libgfortran

Open source component version 4.8.2

Description

The libgfortran contains a Fortran shared library, needed to run Fortran dynamically-linked programs, required for the Vertica LAPACK library addition.

The GNU General Public License (GPL) with Special Library Exception

The author has provided an exception to the GPL allowing the library to be linked with other programs provided that some specific conditions are met. Refer to the license text.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the

original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display

an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or

binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of

software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA

BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Lighttpd Open Source License

Copyright © 2004, Jan Kneschke, incremental

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the 'incremental' nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

lib-javascript-jqplot

Copyright (c) 2009-2010 Chris Leonello

Version 1.0.8

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libio

Version 1.0

Copyright (c) 2013 Joos Kiener <Joos.Kiener@gmail.com>

Description

IO: Classe(s) for improving IO in Java. Currently there is only 1 class in this library.

OptimizedRandomAccessFile: java.io.RandomAccessFile has a readLine() method with terrible performance. It reads files byte per byte and that is very slow. OptimizedRandomAccessFile wraps java.io.RandomAccessFile and exposes all methods while having a readLine() method that performs similar to java.io.BufferedReader and hence about 100 times faster than that of java.io.RandomAccessFile while preserving correct random access.

This software consists of voluntary contributions made by many individuals (AUTHORS.txt, <http://jqueryui.com/about>) For exact contribution history, see the revision history and logs, available at <http://jquery-ui.googlecode.com/svn/>

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libjackson-java

Version 2.3

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the pyOpenSSL package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at these locations:

<https://my.vertica.com/licenses/jackson-core-asl-1.8.8.jar>

<https://my.vertica.com/licenses/jackson-mapper-asl-1.8.8.jar>

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,

MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

libjs-codemirror

Copyright (C) 2014 by Marijn Haverbeke <marijnh@gmail.com> and others

Version 2.0

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libjs-jquery-datatables.columnfilters

Version 1.5.5

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libjs-infovis-toolkit

Copyright © 2013 SenchaLabs - Author: Nicolas Garcia Belmonte

Version 2.0.1

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libjs-jquery-slider

Copyright © 2012 Egor Khmelev

Version 1.1.0

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libjs-jquery-tiptip

Copyright © 2010 Drew Wilson.

Version 1.3

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the

software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term

(regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing

other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

libjs-jquery-ui

Version 1.10.4

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libjs-jquery-ui-multiselect

Copyright (c) 2011 Eric Hynds

Version 1.1.4

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Libopencsv-Java

Copyright © 1999-2012 The Apache Software Foundation. All Rights Reserved.

Version 2.3

Description

opencsv - Library for reading and writing CSV in Java

Opencsv is a very simple csv (comma-separated values) parser library for Java. It supports all the basic csv-type things you're likely to want to do:

- Arbitrary numbers of values per line
- Ignoring commas in quoted elements
- Handling quoted entries with embedded carriage returns (ie entries that span multiple lines)
- Configurable separator and quote characters (or use sensible defaults)
- Read all the entries at once, or use an Iterator style model
- Creating csv files from String[] (ie. automatic escaping of embeddedquote chars)

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution

(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

librdkafka-dev

Version 0.8.6

Copyright (c) 2015

Description

Library implementing the Apache Kafka protocol (development headers), librdkafka is a C implementation of the Apache Kafka protocol. It currently implements the 0.8 version of the protocol and can be used to develop both Producers and Consumers.

More information about Apache Kafka can be found at <http://kafka.apache.org/>

This package contains the development headers.

Declared Ubuntu Licences: BSD-2-clause, BSD-3-clause, MIT

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libsnappy1

Version 1.1.2

Copyright (c) 2015

Description

Snappy (libsnappy1) is a compression/decompression library. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression. For instance, compared to the fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger. On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

Snappy is widely used inside Google, in everything from BigTable and MapReduce to our internal RPC systems. (Snappy has previously been referred to as “Zippy” in some presentations and the likes.)

Copyright 2011, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libtar

Version 1.2.20

Copyright (c) 2015

Description

Tar file manipulation API libtar is a C library for manipulating tar archives. It supports both the strict POSIX tar format and many of the commonly-used GNU extensions.

This software consists of voluntary contributions made by many individuals (AUTHORS.txt, <http://jqueryui.com/about>) For exact contribution history, see the revision history and logs, available at <http://jquery-ui.googlecode.com/svn/>

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libtcmalloc-minimal4

Version 2.0

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Libhttpclient-Java

Copyright © 1999-2012 The Apache Software Foundation. All Rights Reserved

Version 4.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Libhttpcore-Java

Copyright © 2005-2012 The Apache Software Foundation. All Rights Reserved

Version 4.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Libhttpmime-Java

Copyright © 1999-2012 The Apache Software Foundation. All Rights Reserved.

Version 4.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

libuv

Copyright 2015 Joyent, Inc, Node.js is a trademark of Joyent, Inc.

Version 1.0.0

Description

Platform layer for [node.js](#).

libuv is a new platform layer for Node. Its purpose is to abstract IOCP on Windows and libev on Unix systems. We intend to eventually contain all platform differences in this library.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

log4j.log4j

Copyright © 1999-2010 Apache Software Foundation

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

logkit.logkit

Copyright © 2004 Apache Software Foundation.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Lpsolve

Package: lpSolve

Version: 5.6.6

Date: 2011-04-19

Title: Interface to Lp_solve v. 5.5 to solve linear/integer programs

Author: Michel Berkelaar and others

Maintainer: Sam Buttrey <buttrey@nps.edu>

version 5.5.

Description

License: LGPL-2

Packaged: 2011-04-25 22:19:20 UTC; sebuttre

Date/Publication: 2011-04-26 06:30:54

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Lpsolve package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://my.vertica.com/licenses/lpSolve_5.6.6.tar.gz.

Lpsolveapi

Package: lpSolveAPI

Version: 5.5.2.0-5

Date: 2011-07-28

Title: R Interface for lp_solve version 5.5.2.0

Author: lp_solve <<http://lpsolve.sourceforge.net/>>, Kjell Konis <kjell.konis@epfl.ch>.

Maintainer: Kjell Konis <kjell.konis@epfl.ch>

License: LGPL-2

Date/Publication: 2011-08-03 11:41:56

Packaged: 2011-07-28 21:09:07 UTC; rforge

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Lpsolveapi package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://my.vertica.com/licenses/lpSolveAPI_5.5.2.0-5.tar.gz.

Math-Atlas

Open source component version: 3.11.30

Vertica LAPACK library addition. ATLAS (Automatically Tuned Linear Algebra Software) provides optimized Linear Algebra kernels for arbitrary cache-based architectures. ATLAS provides ANSI C and Fortran77 interfaces for the entire BLAS API, and a small portion of the LAPACK AP.

Description

BSD 3-clause "New" or "Revised" License

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

maxent

The Apache Software Foundation.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

MersenneTwister.h

Copyright © 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,

Copyright © 2000 - 2009, Richard J. Wagner

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MIT Kerberos

Version 1.11.3

Copyright © 1985-2013 by the Massachusetts Institute of Technology.

Export of software employing encryption from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original MIT software. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

Individual source code files are copyright MIT, Cygnus Support, Novell, OpenVision Technologies, Oracle, Red Hat, Sun Microsystems, FundsXpress, and others.

Project Athena, Athena, Athena MUSE, Discuss, Hesiod, Kerberos, Moira, and Zephyr are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT.

“Commercial use” means use of a name in a product or other for-profit manner. It does NOT prevent a commercial firm from referring to the MIT trademarks in order to convey information (although in doing so, recognition of their trademark status should be given).

Mockito

Mocking framework for unit tests in Java.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Msgpack

Msgpack is used in Vertica Side Process.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

net.sf.json

Copyright © 2006-2010 Andres Almmiray

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Npm-html2js

Version 0.1.8

Standalone script to load Angular html/jade templates:

<https://www.npmjs.com/package/npm-html2js>

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Oauth

Copyright © 2007, Leah Culver

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

oauth2

Copyright © 2011, Joe Stump

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Open LDAP

Copyright © The OpenLDAP Public License

Version 2.8, 17 August 2003

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

Redistributions in source form must retain copyright statements and notices

Redistributions in binary form must reproduce applicable copyright statements and notices, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution, and

Redistributions must contain a verbatim copy of this document.

The OpenLDAP Foundation may revise this license from time to time. Each revision is distinguished by a version number. You may use this Software under terms of this license revision or under the terms of any subsequent revision of the license.

THIS SOFTWARE IS PROVIDED BY THE OPENLDAP FOUNDATION AND ITS CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENLDAP FOUNDATION, ITS CONTRIBUTORS, OR THE AUTHOR(S) OR OWNER(S) OF THE SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The names of the authors and copyright holders must not be used in advertising or otherwise to promote the sale, use or other dealing in this Software without specific, written prior permission. Title to copyright in this Software shall at all times remain with copyright holders.

OpenLDAP is a registered trademark of the OpenLDAP Foundation.

Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved. Permission to copy and distribute verbatim copies of this document is granted.



Copyright © 2012 The Apache Software Foundation.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

OpenSSL

OpenSSL License

Version 1.0.2b

Copyright © 1998-2016 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"

The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.

Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

org.apache.geronimo.specs.geronimo-annotation_1.0_spec

Copyright © 2003-2011, The Apache Software Foundation

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

org.apache.geronimo.specs.geronimo-jta_1.1_spec

Copyright © 2003-2011, The Apache Software Foundation

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

org.apache.ibatis.ibatis-sqlmap

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written

communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational

purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

org.aspectj.aspectjrt

Copyright © 2011 The Eclipse Foundation. All Rights Reserved.

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the

Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under

this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

org.aspectj.aspectjweaver

Copyright © 2011 The Eclipse Foundation. All Rights Reserved.

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the

Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under

this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

org.codehaus.jackson.jackson-core-asl

Copyright ©2009-2011, FasterXML, LLC

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

org.codehaus.jackson.jackson-mapper- asl

Copyright ©2009-2011 FasterXML, LLC

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

org.eclipse.jdt.core.compiler.ecj

Copyright © 2011 The Eclipse Foundation. All Rights Reserved.

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the

Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under

this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

org.glassfish.web.jstl-impl

Copyright © 2008, 2010 Oracle and/or its affiliates. All rights reserved. Use is subject to license terms.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

org.quartz-scheduler.quartz

Copyright © 2011, Terracotta, Inc.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

orion-ssh2

Copyright © 2010, Juraj Bednar, Trilead AG

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

osgeo-proj.4

PROJ.4 - Cartographic Projections Library, as MIT License (also X11). Vertica uses open-source libraries for geospatial functions.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Paramiko

Copyright © 20011, Robey Pointer

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the paramiko dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at <http://www.my.vertica.com/licenses/paramiko-1.7.7.1.tar.gz>

Paste

Copyright ©, Ian Bicking

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PCRE LICENCE

Copyright © 1997-2010 University of Cambridge

Copyright © 2007-2010, Google Inc.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Perl Artistic License

Copyright © August 15, 1997

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

4. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
 - a. use the modified Package only within your corporation or organization.
 - b. rename any nonstandard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each nonstandard executable that clearly documents how it differs from the Standard Version.
 - c. make other distribution arrangements with the Copyright Holder.
5. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
 - a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
 - b. accompany the distribution with the machine-readable source of the Package with your modifications.
 - c. give nonstandard executables nonstandard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
 - d. make other distribution arrangements with the Copyright Holder.
6. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.
7. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-

called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

8. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.
9. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.
10. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

Pexpect

Copyright © 2010 Noah Spurrier

Credits: Noah Spurrier, Richard Holden, Marco Molteni, Kimberley Burchett, Robert Stone, Hartmut Goebel, Chad Schroeder, Erick Tryzelaar, Dave Kirby, Ids vander Molen, George Todd, Noel Taylor, Nicolas D. Cesar, Alexander Gattin, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, Fernando Perez, Corey Minyard, Jon Cohen, Guillaume Chazarain, Andrew Ryan, Nick Craig-Wood, Andrew Stone, Jorgen Grahn (Let me know if I forgot anyone.)

Free, open source, and all that good stuff.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Phantomjs

PhantomJS (phantomjs.org) is a headless WebKit, capable of scripting with JavaScript.

LICENSE.BSD

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PHP License

Copyright © The PHP License, version 3.01

Copyright © 1999 - 2009 The PHP Group. All rights reserved.

1. Redistribution and use in source and binary forms, with or without modification, is permitted provided that the following conditions are met:
2. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
3. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
4. The name "PHP" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact group@php.net.
5. Products derived from this software may not be called "PHP", nor may "PHP" appear in their name, without prior written permission from group@php.net. You may indicate that your software works in conjunction with PHP by saying "Foo for PHP" instead of calling it "PHP Foo" or "phpfoo"
6. The PHP Group may publish revised and/or new versions of the license from time to time. Each version will be given a distinguishing version number.

Once covered code has been published under a particular version of the license, you may always continue to use it under the terms of that version. You may also choose to use such covered code under the terms of any subsequent version of the license published by the PHP Group. No one other than the PHP Group has the right to modify the terms applicable to covered code created under this License.

Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes PHP software, freely available from <http://www.php.net/software/>".

THIS SOFTWARE IS PROVIDED BY THE PHP DEVELOPMENT TEAM ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PHP DEVELOPMENT TEAM OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the PHP Group.

The PHP Group can be contacted via Email at group@php.net.

For more information on the PHP Group and the PHP project, please see [<http://www.php.net>](http://www.php.net).

PHP includes the Zend Engine, freely available at [<http://www.zend.com>](http://www.zend.com).

PostgreSQL

This product uses the PostgreSQL Database Management System (formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2005, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

protobuf

Copyright 2008 Google Inc. All rights reserved.

Version 2.4.1, 2.6.0

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

protobuf-java

Copyright 2008 Google Inc. All rights reserved.

Version 2.4.1

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Pybonjour

Copyright ©, cstawarz@gmail.com

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Pyopenssl

Copyright ©, Jean-Paul Calderone

pyOpenSSL 0.11 is distributed under the GNU Lesser General Public License, below.

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the pyOpenSSL package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at <http://www.my.vertica.com/licenses/pyOpenSSL-0.11.tar.gz>.

PySNMP

Copyright © 1999-2012 Ilya Etingof

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Python

Copyright © 2001- 2012 Python Software Foundation; All Rights Reserved

This is the official license for the Python 2.7.* release.

Description

Interactive high-level object-oriented language (default version).

Python, the high-level, interactive object oriented language, includes an extensive class library with lots of goodies for network programming, system administration, sounds and graphics.

This package is a dependency package, which depends on Debian's default Python version (currently v2.7).

History of the Software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <http://www.cwi.nl/>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <http://www.cnri.reston.va.us/>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see <http://www.zope.com/>). In 2001, the Python Software Foundation (PSF, see <http://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <http://www.opensource.org/> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL-compatible? (1)
---------	--------------	------	-------	---------------------

0.9.0 thru 1.2		1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	yes (2)
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2002-2003	PSF	yes
2.3.3	2.3.2	2002-2003	PSF	yes
2.3.4	2.3.3	2004	PSF	yes
2.3.5	2.3.4	2005	PSF	yes
2.4	2.3	2004	PSF	yes
2.4.1	2.4	2005	PSF	yes
2.4.2	2.4.1	2005	PSF	yes
2.4.3	2.4.2	2006	PSF	yes
2.5	2.4	2006	PSF	yes
2.7	2.6	2010	PSF	yes

Note: 1. GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

2. According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

Terms and Conditions for Accessing or Otherwise Using Python

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.
4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com (“BeOpen”), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization (“Licensee”) accessing and otherwise using this software in source or binary form and its associated documentation (“the Software”).
2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
3. BeOpen is making the Software available to Licensee on an “AS IS” basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the “BeOpen Python”

logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (“CNRI”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI’s License Agreement and CNRI’s notice of copyright, i.e., “Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved” are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI’s License Agreement, Licensee may substitute the following text (omitting the quotes): “Python 1.6.1 is made available subject to the terms and conditions in CNRI’s License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>.”
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
4. CNRI is making Python 1.6.1 available to Licensee on an “AS IS” basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Python3

A. HISTORY OF THE SOFTWARE

=====

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <http://www.cwi.nl>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <http://www.cnri.reston.va.us>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see <http://www.zope.com>). In 2001, the Python Software Foundation (PSF, see <http://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <http://www.opensource.org> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release Derived Year Owner GPL-from compatible? (1)

0.9.0 thru 1.2	1991-1995	CWI	yes	
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	yes (2)
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2 and above	2.1.1	2001-now	PSF	yes

Footnotes:

(1) GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with othersoftware that is released under the GPL; the others don't.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON

=====

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at

<http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1

alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam,
The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Python Dialog

The Administration Tools part of this product uses Python Dialog, a Python module for doing console-mode user interaction.

Upstream Author:

Peter Astrand <peter@cendio.se>

Robb Shecter <robb@acm.org>

Sultanbek Tezadov

Florent Rougon <flo@via.ecp.fr>

Copyright © 2000 Robb Shecter, Sultanbek Tezadov

Copyright © 2002, 2003, 2004 Florent Rougon

License Description

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Python dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at <http://www.my.vertica.com/licenses/pythondialog-2.7.tar.bz2>

Python Pip

Copyright © 2008- 2016 The pip developers (see [AUTHORS.txt](#)).

Description

The Python pip module is for installing packages.

Installing the Vertica server RPM, includes the python distribution. Python is installed under the open source software (oss) directory, `/opt/vertica/oss/python`. A sub-directory of the python directory includes the pip module.

Copyright (c) 2008-2016 The pip developers (see `AUTHORS.txt` file)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

AUTHORS File Contents

For the most recent version of `AUTHORS.txt`, see <https://github.com/pypa/pip/blob/develop/AUTHORS.txt>.

Quartz

Copyright © 2001-2010 Terracotta, Inc.

Version 1.6.1

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of

this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or

malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Quartz-scheduler-quartz

Code for Quartz scheduler

Version 1.8.5

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of

this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or

malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

QUnit

Copyright 2013 The JQuery Foundation

All rights reserved.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Qunitjs

QUnit is a powerful, easy-to-use JavaScript unit testing framework.

All rights reserved.

Description

Copyright jQuery Foundation and other contributors, <https://jquery.org/>

This software consists of voluntary contributions made by many individuals. For exact contribution history, see the revision history available at <https://github.com/jquery/qunitjs.com>.

The following license applies to all parts of this software except as documented below:

====

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

====

Copyright and related rights for sample code are waived via CC0. Sample code is defined as all source code displayed within the prose of the documentation and all examples and demos.

CC0: <http://creativecommons.org/publicdomain/zero/1.0/>

====

All files located in the `node_modules` directory are externally maintained libraries used by this software which have their own licenses; we recommend you read them, as their terms may differ from the terms above.

R

R version 3.0.0 (2013-04-03) -- "Masked Marvel"

Copyright (C) 2013 The R Foundation for Statistical Computing

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at nocharge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to

the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for

exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights

that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

The complete source code of the R package can be found at <https://my.vertica.com/licenses/R-2.14.0.tar.gz>.

Sources for the Vertica User Defined Functions in R can be found at <https://my.vertica.com/downloads/>.

r-cran-rcpp

Copyright © Dirk Eddelbuettel and Romain Francois, with contributions by Douglas Bates and John Chambers

Version 0.9.5

Description

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and

control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at nocharge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to

the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for

exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights

that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

You can download source code of the r-cran-rcpp package from:

https://my.vertica.com/licenses/Rcpp_0.9.5.tar.gz.

rollingfilesink.java

Copyright © 2012 The Apache Software Foundation.

Description

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Rinside

Copyright © Dirk Eddelbuettel and Romain Francois, with contributions by Douglas Bates and John Chambers

Version 0.2.4

Description

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The complete source code of the rinside package can be found at

RRDTool Open Source License

Copyright © RRDTool Open Source License

Note: rrdtool is a dependency of using the ganglia-web third-party tool. RRDTool allows the graphs displayed by ganglia-web to be produced.

RRDTOOL - Round Robin Database Tool

A tool for fast logging of numerical data graphical display of this data.

Copyright © 1998-2008 Tobias Oetiker

All rights reserved.

Description

GNU GPL License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

FLOSS License Exception

(Adapted from <http://www.mysql.com/company/legal/licensing/foss-exception.html>)

I want specified Free/Libre and Open Source Software ("FLOSS") applications to be able to use specified GPL-licensed RRDtool libraries (the "Program") despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the "GPL").

As a special exception to the terms and conditions of version 2.0 of the GPL:

You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a "FLOSS Work") licensed under one or more of the licenses listed below, as long as:

1. You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves
2. All identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves
 - are distributed subject to one of the FLOSS licenses listed below, and
 - the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections.
3. Any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL.

FLOSS License List

License name Version(s)/Copyright Date

Academic Free License 2.0

Apache Software License 1.0/1.1/2.0

Apple Public Source License 2.0

Artistic license From Perl 5.8.0

BSD license "July 22 1999"

Common Public License 1.0

GNU Library or "Lesser" General Public License (LGPL) 2.0/2.1

IBM Public License, Version 1.0

Jabber Open Source License 1.0

MIT License (As listed in file MIT-License.txt) -

Mozilla Public License (MPL) 1.0/1.1

Open Software License 2.0

OpenSSL license (with original SSLeay license) "2003" ("1998")

PHP License 3.0

Python license (CNRI Python License) -

Python Software Foundation License 2.1.1

Sleepycat License "1999"

W3C License "2001"

X11 License "2001"

Zlib/libpng License -

Zope Public License 2.0/2.1

rsync

rsync was originally written by Andrew Tridgell and is currently maintained by Wayne Davison. It has been improved by many developers from around the world.

Description

rsync may be used, modified and redistributed only under the terms of the GNU General Public License, found at:

<http://www.fsf.org/licenses/gpl.html>

The GNU General Public License applies just to rsync and no other components of Vertica.

You are entitled to receive the source code for such software. For no less than three years from the date you obtained this software package, you may download a copy of the source code for the software in this package licensed under the GPL at no charge by visiting:

<http://www.my.vertica.com/licenses/rsync-3.0.7.tar.gz>

http://my.vertica.com/licenses/vertica_rsync_patch.txt

You can download this source code so it remains separate from other software on your computer system.

scons

Copyright (c) 2004-2011 All rights reserved

Version 2.2.0

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

selenium-htmlunit-driver

Copyright © 2012 The Apache Software Foundation.

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Setuptools

Copyright © 1990 - 2011, Philip J. Eby

Zope Public License (ZPL) Version 2.0

This software is Copyright (c) Zope Corporation (tm) and Contributors. All rights reserved.

This license has been certified as open source. It has also been designated as GPL compatible by the Free Software Foundation (FSF).

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions in source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name Zope Corporation (tm) must not be used to endorse or promote products derived from this software without prior written permission from Zope Corporation.
4. The right to distribute this software or to use it for any purpose does not give you the right to use Servicemarks (sm) or Trademarks (tm) of Zope Corporation. Use of them is covered in a separate agreement (see <http://www.zope.com/Marks>).
5. If any files are modified, you must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

Disclaimer

THIS SOFTWARE IS PROVIDED BY ZOPE CORPORATION ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ZOPE CORPORATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of contributions made by Zope Corporation and many individuals on behalf of Zope Corporation. Specific attributions are listed in the accompanying credits file.

signpost-commonshhttp4

Copyright 2012 Matthias Käppler

Version 1.2.1.1

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Signpost-Core

Copyright 2012 Matthias Käppler

Version 1.2.1.1

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Signpost-Commonshttp 4 1.2.1.1

Copyright © 2012 Matthias Käppler

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

slf4j

Copyright (c) 2004-2013 QOS.ch

Version 1.7.5

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

slf4j-log4j12

Copyright (c) 2004-2013 QOS.ch

Version 1.7.5

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

slf4j-simple

Copyright (c) 2004-2011 All rights reserved

Version 1.6.4

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Spring-Ldap

Copyright © 2005-2010 Mattias Arthursson, Ulrik Sandberg, Eric Dalquist, Keith Barlow

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Spring MVC

Copyright © 2011, Spring Source.org

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

springframework-security 3.2

Copyright © 2004-2010 Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Alef Arendsen, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma, Chris Beams, Tareq Abedrabbo, Andy Clement, Dave Syer, Oliver Gierke

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Springframework-Tx 3.0.5

Copyright © 2004-2010 Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Alef Arendsen, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma, Chris Beams, Tareq Abedrabbo, Andy Clement, Dave Syer, Oliver Gierke

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within

such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

springframework-webmvc

Version 4.0.0

Copyright © 2012

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

spring-mobile

Copyright © 2012

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

SNMP

Various copyrights apply to this package, listed in various separate parts below. Please make sure that you read all the parts. Up until 2001, the project was based at UC Davis, and the first part covers all code written during this time. From 2001 onwards, the project has been based at SourceForge, and Networks Associates Technology, Inc hold the copyright on behalf of the wider Net-SNMP community, covering all derivative work done since then. An additional copyright section has been added as Part 3 below also under a BSD license for the work contributed by Cambridge Broadband Ltd. to the project since 2001. An additional copyright section has been added as Part 4 below also under a BSD license for the work contributed by Sun Microsystems, Inc. to the project since 2003.

Code has been contributed to this project by many people over the years it has been in development, and a full list of contributors can be found in the README file under the THANKS section.

Part 1: CMU/UCD copyright notice: (BSD like)

Copyright © 1989, 1991, 1992 by Carnegie Mellon University

Derivative Work - 1996, 1998-2000

Copyright © 1996, 1998-2000 The Regents of the University of California

All Rights Reserved

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of CMU and The Regents of the University of California not be used in advertising or publicity pertaining to distribution of the software without specific written permission.

CMU AND THE REGENTS OF THE UNIVERSITY OF CALIFORNIA DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CMU OR THE REGENTS OF THE UNIVERSITY OF CALIFORNIA BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM THE LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Part 2: Networks Associates Technology, Inc copyright notice (BSD)

Copyright © 2001-2003, Networks Associates Technology, Inc

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Networks Associates Technology, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 3: Cambridge Broadband Ltd. copyright notice (BSD)

Portions of this code are copyright (c) 2001-2003, Cambridge Broadband Ltd.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of Cambridge Broadband Ltd. may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 4: Sun Microsystems, Inc. copyright notice (BSD)

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Use is subject to license terms below.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 5: Sparta, Inc copyright notice (BSD)

Copyright © 2003-2006, Sparta, Inc

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Sparta, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 6: Cisco/BUPTNIC copyright notice (BSD)

Copyright © 2004, Cisco, Inc and Information Network Center of Beijing University of Posts and Telecommunications.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Cisco, Inc, Beijing University of Posts and Telecommunications, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 7: Fabasoft R&D Software GmbH & Co KG copyright notice (BSD)

Copyright © Fabasoft R&D Software GmbH & Co KG, 2003

oss@fabasoft.com

Author: Bernhard Penz

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of Fabasoft R&D Software GmbH & Co KG or any of its subsidiaries, brand or product names may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE

LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Standard

Copyright 2004-2012 Apache Software Foundation

Version 1.1.2

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is

intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Tecla Command-Line Editing

Copyright © 2000 by Martin C. Shepherd.

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

twittersource.java

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written

communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational

purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

twittersourceconstants.java

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written

communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational

purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

UI-Grid

MC uses UI-Grid v3.0.x for Data Ingest and Hadoop tables.

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

WebOb

Copyright © 2007 Ian Bicking, Sergey Schetinin

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Webware

Copyright © Ian Bicking and Contributors

History of the Software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <http://www.cwi.nl/>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <http://www.cnri.reston.va.us/>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see <http://www.zope.com/>). In 2001, the Python Software Foundation (PSF, see <http://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <http://www.opensource.org/> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL-compatible? (1)
0.9.0 thru 1.2		1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	yes (2)
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2002-2003	PSF	yes
2.3.3	2.3.2	2002-2003	PSF	yes

2.3.4	2.3.3	2004	PSF	yes
2.3.5	2.3.4	2005	PSF	yes
2.4	2.3	2004	PSF	yes
2.4.1	2.4	2005	PSF	yes
2.4.2	2.4.1	2005	PSF	yes
2.4.3	2.4.2	2006	PSF	yes
2.5	2.4	2006	PSF	yes
2.7	2.6	2010	PSF	yes

Note: 1. GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

2. According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

Terms and Conditions for Accessing or Otherwise Using Python

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided

herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").
2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the

BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an “AS IS” basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the “BeOpen Python” logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.
7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (“CNRI”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use

Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>."

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant

permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the “ACCEPT” button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.



Copyright 2006-1015 Microsoft

Version 3.9

Microsoft Reciprocal License (Ms-RL)

Description

WiX Toolset License

The WiX toolset is released under the Microsoft Reciprocal License (MS-RL). A reciprocal license is used to ensure that others who build on the effort of the WiX community give back to the WiX community. Specifically the license changes and improvements to the WiX toolset must be published using the same license.

Sometimes the reciprocal license is incorrectly interpreted to also apply to bundles, packages, custom actions built using the WiX toolset. The Outercurve Foundation has provided this statement to clarify:

The WiX toolset (WiX) is licensed under the Microsoft Reciprocal License (MS-RL). The MS-RL governs the distribution of the software licensed under it, as well as derivative works, and incorporates the definition of a derivative work provided in U.S. copyright law. OuterCurve Foundation does not view the installer packages generated by WiX as falling within the definition of a derivative work, merely because they are produced using WiX. Thus, the installer packages generated by WiX will normally fall outside the scope of the MS-RL, and any of your source code, binaries, libraries, routines or other software components that are incorporated in installer packages generated by WiX can be governed by other licensing terms.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS

FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Xdan-datetimepicker

Version 2.5.3

JQuery plugin for some drop down lists in Vertica Management Console (MC).

MIT License (MIT)

Copyright (c) 2016

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Xerces

NOTICE file corresponding to section 4(d) of the Apache License, Version 2.0, in this case for the Apache Xerces distribution.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Portions of this software were originally based on the following: Software copyright © 1999, IBM Corporation., <http://www.ibm.com>.

xml-commons

Copyright © 2012

<http://www.apache.org/licenses/>

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the

purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS



Copyright (c) 2007-2011, Lloyd Hilaiel

<http://github.com/lloyd/yajl>

Copyright © 2004-2013 by Internet Systems Consortium, Inc. ("ISC")

Copyright © 1995-2003 by Internet Software Consortium

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Zlib

This is used by the project to load zipped files directly by COPY command. www.zlib.net/
zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.3, July 18th, 2005
Copyright © 1995-2005 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

- The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org

Mark Adler madler@alumni.caltech.edu

Jean-loup Gailly jloup@gzip.org

Mark Adler madler@alumni.caltech.edu